High-performance hardware accelerators for image processing in space applications

POLITECNICO DI TORINO

SCUOLA INTERPOLITECNICA DI DOTTORATO

Doctoral Program in Computer and Control Engineering

Final Dissertation

High-performance hardware accelerators for image processing in space applications

Daniele ROLFO

Tutor
prof. Paolo PRINETTO

Co-ordinator of the Research Doctorate Course
prof. Matteo SONZA REORDA

05/03/2015

Attachment n° 5

SELECTION PANEL

On March 5, 2015 Mr. D. Rolfo has defended in the Department of Control and Computer Engineering of the Politecnico di Torino, a thesis with the following title: High-performance hardware accelerators for image processing in space applications.

The methodology was assessed as solid.

The research yielded interesting results and showed excellent critical thinking skills.

During the interview the candidate showed a commanding knowledge and understanding of the problems dealt with.

The Board unanimously assessed as well qualified the research work and decided to award the **degree of High Qualified Research Doctor** to the candidate.

Prof. M. Gerla (President)

Prof. C. Metra (Member)

Prof. R. De Mori (Secretary)

ACKNOWLEDGEMENTS

It is a pleasure to thank my tutor Prof.Paolo PRINETTO. Thanks to his patience, motivation, enthusiasm, and immense knowledge I grown a lot not just from the professional point of view. In the same way, I really appreciated the help and the guidelines provided by Prof. Stefano DI CARLO. Without them it would not be possible to reach the results that my research work had.

I would like to gratefully acknowledge all the industrial partners that collaborated with me in these years. I want to mention the guys from *Thales Alenia Space Italy*, in particular Ing. Andrea MARTELLI, Ing. Antonio TRAMUTOLA, and Ing. Piergiorgio LANZA, that during the numerous meetings provided me a lot of guidelines, and ideas that allowed to increase the relevance of my research activity.

Moreover, I want to thanks Ing. Girgio MAGISTRATI, Ing. Gianluca FURANO, and Ing. Marco ROVATTI that allowed me to spend six months as scientist visitor at the *European Space and Technology Centre*, during whose I conducted a really exciting research work.

My sincere thanks also goes to all guys of LAB6, in particular to Alessandro (Spitfire), Alessandro (MtR), Salvatore, and Marco, who supported me for the last three years.

I want to separately mention other two guys from LAB6: Giulio and Pascal.

Many thanks go to Giulio who supported and stood me during the last 14 years. The best study and work partner I could have.

It is a pleasure to mention Pascal, I will never forget the time spent with him and in particular the "creiamo intelligenza" brain-storming we did in the last years.

Then, how could I forget Davide and Diego, two guys that had my same "sick" idea to do a PhD. I will forever remember the "revival" dinners and lunches that we had in these three years.

I am grateful to all my friends for their continued moral support.

Eventually, I am forever indebted to my parents, Gianni and Ada, and my brother, Claudio, for their understanding, endless patience and encouragement when it was most required.

And last but not least Giulia, for the very special person she is and for the incredible amount of patience she had with me in the last three years.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Mars is a hard place to reach. While there have been many notable success stories in getting probes to the Red Planet, the historical record is full of bad news.
Counting all Soviet/Russian, U.S., European, and Japanese attempts, more than half of Mars missions have failed. Just to mention some examples, *Korabl 11*, *Korabl 13*, *Kosmos 419*, *Phobos 1 Orbiter*, *Mars Observer*, and *Phobos-Grunt/Yinghuo-1* represent the main missions failed during the trip to Mars because of collisions with space debris in orbit around Earth.

The success rate for actually landing on the Martian surface is even worse, roughly 30 percent. This low success rate must be mainly credited to the Mars environment characteristics. In the Mars atmosphere strong winds frequently breath. This phenomena usually modifies the lander descending trajectory diverging it from the target one. Moreover, the Mars surface is not the best place where performing a safe land. In fact, it is pitched by many and close craters and huge stones, and characterized by huge mountains and hills (e.g., *Olympus Mons* is 648 kilometres in diameter and 27 kilometres tall). For these reasons a mission failure due to a landing in huge craters, on big stones or on part of the surface characterized by a high slope is highly probable. Some examples of these failures are the landers of *Mars 2*, *Mars 6*, *Mars Polar Lander*, and *Beagle 2*.

In the last years, all space agencies have increased their research efforts in order to enhance the success rate of Mars missions. In particular, the two hottest research topics are: the active debris removal and the guided landing on Mars.

The former aims at finding new methods to remove space debris exploiting unmanned spacecrafts. These must be able to autonomously: detect a debris, analyse it, in order to extract its characteristics in terms of weight, speed and dimension, and, eventually, rendezvous with it. In order to perform these tasks, the spacecraft must have high vision capabilities. In other words, it must be able to take pictures and process them with very complex image processing algorithms in order to detect, track and analyse the debris.

The latter aims at increasing the landing point precision (i.e., landing ellipse) on Mars. Fu-

ture space-missions will increasingly adopt Video Based Navigation systems to assist the entry, descent and landing (EDL) phase of space modules (e.g., spacecrafts), enhancing the precision of automatic EDL navigation systems. For instance, recent space exploration missions, e.g., *Spirity*, *Oppurtunity*, and *Curiosity*, made use of an EDL procedure aiming at following a fixed and precomputed descending trajectory to reach a precise landing point. This approach guarantees a maximum landing point precision of 20 km. By comparing this data with the Mars environment characteristics, it is possible to understand how the mission failure probability still remain really high.
A very challenging problem is to design an autonomous-guided EDL system able to even more reduce the landing ellipse, guaranteeing to avoid the landing in dangerous area of Mars surface (e.g., huge craters or big stones) that could lead to the mission failure. The autonomous behaviour of the system is mandatory since a manual driven approach is not feasible due to the distance between Earth and Mars. Since this distance varies from 56 to 100 millions of kilometres approximately due to the orbit eccentricity, even if a signal transmission at the light speed could be possible, in the best case the transmission time would be around 31 minutes, exceeding so the overall duration of the EDL phase.

In both applications, all the algorithms must guarantee self-adaptability to the environmental conditions. Since the Mars (and in general the space) harsh conditions are difficult to be predicted at design time, these algorithms must be able to automatically tune the internal parameters depending on the current conditions.

Moreover, real-time performances is another key factor. Since a software implementation of these computational intensive tasks cannot reach the required performances, these algorithms must be accelerated via hardware.

For this reasons, this thesis presents my research work done on advanced image processing algorithms for space applications and the associated hardware accelerators. My research activity has been focused on both the algorithm and their hardware implementations. Concerning the first aspect, I mainly focused my research effort to integrate self-adaptability features in the existing algorithms. While concerning the second, I studied and validated a methodology to efficiently develop, verify and validate hardware components aimed at accelerating video-based applications. This approach allowed me to develop and test high performance hardware accelerators that strongly overcome the performances of the actual state-of-the-art implementations.

In addition to this introduction chapter, the sequel of the thesis is organized in four main chapters.

Chapter 2 starts with a brief introduction about the story of digital image processing, starting from the first picture taken in space to the current sophisticated image processing systems. However, the main content of this chapter is the description of space missions in which digital image processing has a key role. A major effort has been spent on the missions in which my research activity has a substantial impact. In particular, for these missions, this chapter deeply analizes

and evaluates the state-of-the-art approaches and algorithms. The reported information allow the reader to become more familiar with the image processing algorithms applied in space and to understand their level of complexity.

Chapter 3 analyzes and compares the two technologies used to implement high performances hardware accelerators, i.e., *Application Specific Integrated Circuits* (ASICs) and *Field Programmable Gate Arraies* (FPGAs). Thanks to this information the reader may understand the main reasons behind the decision of space agencies to exploit FPGAs instead of ASICs for high-performance hardware accelerators in space missions, even if FPGAs are more sensible to *Single Event Upsets* (i.e., transient error induced on hardware component by alpha particles and solar radiation in space). Moreover, this chapter deeply describes the three available space-grade FPGA technologies: *One-time Programmable*, *Flash-based*, and *SRAM-based*. For each of them a comprehensive explanation of their internal architecture is reported, highlighting the pros and cons of each technology. In order to better understand the differences among these technologies, a comparison in terms of speed, available resources and robustness is reported. Eventually, this chapter presents the main fault-mitigation techniques against SEUs that are mandatory for employing space-grade FPGAs in actual missions.

Chapter 4 describes one of the main contribution of my research work: a library of high-performance hardware accelerators for image processing in space applications. The basic idea behind this library is to offer to designers a set of validated hardware components able to strongly speed up the basic image processing operations commonly used in an image processing chain. In other words, these components can be directly used as elementary building blocks to easily create a complex image processing system, without wasting time in the debug and validation phase. It must be mentioned that, during the development of these components, the design effort has not just focused on the achieved performances in terms of resources usage and speed, but also in the integration of self-adaptability features to the environmental conditions.
This library groups the proposed hardware accelerators in IP-core families. The components contained in a same family share the same provided functionality and input/output interface. This harmonization in the I/O interface enables to substitute, inside a complex image processing system, components of the same family without requiring modifications to the system communication infrastructure. However, in addition to the analysis of the internal architecture of the proposed components, another important aspect of this chapter is the methodology used to develop, verify and validate the proposed high performance image processing hardware accelerators. This methodology, that can be freely applied in the design and validation of every kind of hardware accelerators, involves the usage of different programming and hardware description languages in order to support the designer from the algorithm modelling up to the hardware implementation and validation.

Chapter 5 presents the proposed complex image processing systems. In particular, it exploits a set of actual case studies, associated with the most recent space agencies needs, to show how

the hardware accelerator components can be assembled to build a complex image processing system. In addition to the hardware accelerators contained in the library, the described complex system embeds innovative ad-hoc hardware components and software routines able to provide high performance and self-adaptable image processing functionalities. To prove the benefits of the proposed methodology, each case study is concluded with a comparison with the current state-of-the-art implementations, highlighting the benefits in terms of performances and self-adaptability to the environmental conditions.

CHAPTER 2

DIGITAL IMAGE PROCESSING: A LONG HISTORY TO REACH SPACE APPLICATIONS

Photography has a long history started in 1826 when Joseph Nicephore Niepce created a photomechanical reproduction of an engraving of the Cardinal Georges d'Amboise through *heliography* process (Figure 2.1) [24]. This can be considered the first photo taken in the world.

Figure 2.1: Photoetching of an engraving of Cardinal Georges D'Amboise

Since the heliography process requires up to 8 hours of exposure to provide an acceptable result, Louis Daguerre, the associate of Joseph Nicephore Niepce, in 1839 invented the *metal-based daguerreotype* process (Figure 2.2) [24]. This process provides excellent quality pictures, for that period, requiring an exposure time of just one minute.
The metal-based daguerreotype process soon had some competition from the paper-based calotype negative and salt print processes invented by Henry Fox Talbot [12]. Subsequent innovations reduced the required camera exposure time from minutes to seconds and eventually to a small fraction of a second; the introduced new photographic media were more economical, sensitive or

Figure 2.2: Edgard Allan Poe daguerrotype picture

convenient, including roll films for casual use by amateurs; and made it possible to take pictures in natural color as well as in black-and-white.
Roll film dominated the early years of photography. In the 1880's George Eastman [115] founded *Kodak* and made photography available to all persons. Photography attracted virtually everyone on the planet, and a vast variety of cameras were created, from *Brownies* [116] and disposables to *Hasselblads* [88] and *Leicas* [167]. A look at the fantastic colours permanently captured in 70-year-old Kodachrome images (Figure 2.3) leads easily and correctly to the conclusion that film is a near-perfect medium for the storage of images.

Figure 2.3: Kodachrome image example

The first picture taken is space environment date back to 1964, when *Ranger 7* [159] (i.e., the first fully successful mission of the *Ranger Program* aiming at reaching the Moon surface) sent the first picture, taken at an altitude of 2,110 km, on 31st of July 1964 at 13:08:45 UT. Then, during the descending phase, it took and successfully sent to Earth 4,308 high resolution images. *Ranger 7* was not equipped with standard roll film cameras, but with more sophisticated *Vidicon* cameras

[174]. The main components of these cameras was the *Vidicon tube* (Figure 2.4, a cathode ray tube in which the cathode ray was scanned across a target which was illuminated by the scene to be taken).

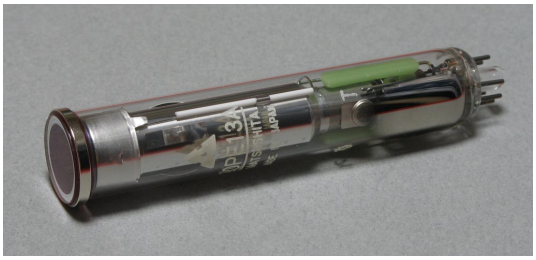


Figure 2.4: *Vidicon* tube

In particular, *Ranger 7* was equipped with six *Vidicon* cameras (2 wide angle and 4 narrow angle cameras) organized in two self-contained channels, to reach the greatest reliability and probability of obtaining high-quality video pictures. Figure 2.5 shows the organization of the cameras system in the *Ranger 7* lander.

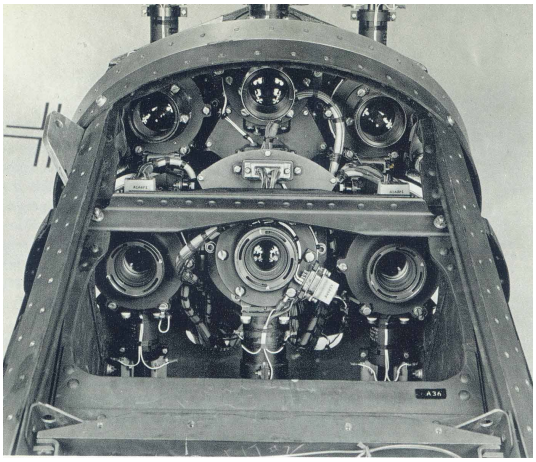


Figure 2.5: *Ranger 7* camera system

Hasselblad from 1966 started to develop a group of roll film-based camera suitable for space applications [89]. This development process had the most important success in the *Apollo 11* mission [149], in which the *Hasselblad 500EL/M*(Figure 2.6) was used by Neil Armstrong to capture the first human walk on the Moon surface.

The commercial introduction of computer-based electronic digital cameras in the 1990s soon revolutionized photography. During the first decade of the 21st century, traditional film-based photochemical methods were increasingly marginalized as the practical advantages of the new

Figure 2.6: *Hasselblad 500EL/M* camera

technology became widely appreciated and the image quality of moderately priced digital cameras was continually improved.
This improvement in quality can be appreciated in the digital space camera, as well. In fact, analysing pictures taken by modern digital cameras involved in space missions (Figure 2.7) the high quality and the fidelity of the captured scenes became evident.

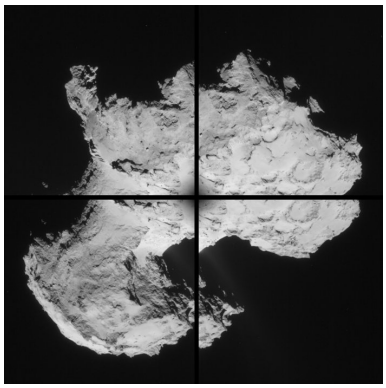
In addition to the increase of quality, digital images introduced the possibility to perform the so called *Digital Image Processing* (DIP). Basically, DIP manipulates digital images to enhance the image quality or to extract useful information from the processed image.
DIP operations can be grouped into seven different steps:

1. ***Image Acquisition*** aims at acquiring a digital image. In other words, the scene in front of the camera is first sampled in the spatial domain, and then discretized in the brightness domain. In this way the scene is translated in a 2D matrix composed of a set of rows and columns of pixels (i.e., the basic element composing a digital image). The number of rows and columns, that defines the *image resolution*, depends on the spatial domain sampling. The range of possible value of each pixel, that defines the *pixel resolution*, is related to the adopted brightness quantization.

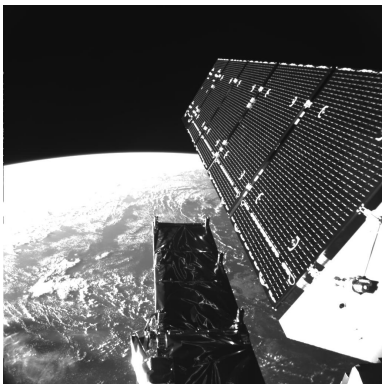
2. ***Image preprocessing*** improves the image quality to increase the accuracy of successive processing steps. There are a plenty of image preprocessing operations, examples are *image filtering* [104] and *image enhancement* [130]. There is not a fixed sequence of preprocessing operations able to provide the best results, but the operations to be applied must be properly select depending on the next image processing steps.

3. ***Image segmentation*** aims at partitioning an input image into regions. It enables to ease the DIP problem focusing the processing effort on segments of more interest, only.

4. ***Image representation*** converts digital images in a format suitable for computer processing. It usually adds some information to the 2D matrix composing the input image, such as

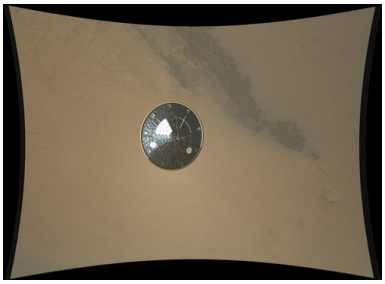


(a) comet *67P/Churyumov-Gerasimenko* taken by *NAVigation CAMera* (NAVCAM) [161] during *Rosetta* mission [7]



(b) Deployed solar wings with the Radar open below of the *Sentinel 1* orbiter [66] taken by the *Digital Space Micro-Camera* [181]

(c) First high-resolution color mosaic from NASA Curiosity rover [150] taken by the *Mastcam* [158]



(d) Curiosity's Heat Shield [150] taken during the descending phase on Mars by the *MARs Descent Imager* (MARDI) [157]

Figure 2.7: Examples of picture taken by modern digital space cameras

the number of rows and columns, and the bit used to represent each pixel (i.e., *bit-per-pixel*(bpp) resolution), to allow computers to properly read the image content. There are a plenty of different image formats (for an exhaustive list of these formats you can refer to [220]).

5. ***Image registration*** aims at extracting features that result in some quantitative information of interest, or features, that are basic for differentiating one class of objects from another. It is also called *dimensional reduction* since it reduces the information contained in the image, allowing to focus the processing effort of the consecutive steps just on the extracted features. The features extraction methods can be of different types, going from the low-level ones (e.g., *Edge detectors* [46], *Corner detectors* [30], etc.) to the *shape-based* one (e.g., *Blob detectors* [44], *Template matcher* [15], etc.).

6. ***Image recognition*** merges information extracted from the image registration and segmen-

tation to assign a label to an image region based on the information provided by its descriptors. Pixels belonging to the same region has a common characteristic, so they can be labelled in the same way. For this reason image recognition is also called *image labelling* [178]. The main goal of this operation is to locate object and boundaries in the image, like in medicine to identify tumours or other pathologies, or in video surveillance for pedestrian and car detection.

7. ***Image interpretation*** aims at assigning a meaning to an ensemble of recognized objects or features. This final step is always performed at the end of the image processing chain, in order to analyse the information extracted from the previous image processing steps.

Differently from human visual system, DIP can work on different wavelength domains: from Infra-red to visible light, and from Ultraviolet ray to X-rays. However, in this thesis only the visible light will be taken into account, since the presented research works are related to the processing of images created from the reflection of visible light.

In the last years, DIP has been deeply used in space applications. The main applications in which DIP has a key role are:

- *Earth observation*
- *Space cartography*
- *Satellite attitude control*
- *Rover navigation*
- *Video-Based Navigation (VBN) landing system*
- *Active space debris removal*
- *Image compression*

Next sections provide an overview on the importance of DIP in the reported space applications. More detailed information are provided for the last three applications (i.e., VBN landing system, Active space debris removal, and Image compression) since my research activity has been mainly focused on them.

2.1 Earth observation

Earth observation is the gathering of information about planet Earth's physical, chemical and biological systems via remote sensing technologies supplemented by earth surveying techniques, encompassing the collection, analysis and presentation of data [65]. Its main purpose is a scientific study of the Earth and human life evolution. However, nowadays it is becoming more

and more important due to the dramatic impact that modern human civilization is having on the Earth, and the need to minimize negative impacts along with the opportunities Earth observation provides to improve social and economic well-being.

Earth observation is one of the most important application of DIP in space. In fact, during an Earth observation mission, an artificial satellite revolving around the earth takes a set of pictures that must be first pre-processed to remove haze, cloud and sensor induced defects, and then must be processed to automatically extract useful information concerning the Earth and human life evolution. These information are extracted exploiting all the potentiality of DIP since used sensors exploits the full light spectrum.

According to the NASA classification [160], Earth observation can be split in five macro-areas, depending on the application target:

- *Atmosphere*: it envelops all the applications concerning the Earth atmosphere monitoring. One of the most important monitoring action concerns the *Carbon Monoxide* concentration in the atmosphere. This is of particular importance because this gas is causing global warming and air pollution.
  Another relevant monitoring concerns clouds. In fact, clouds have the ability to warm and cool the Earth surface. So their analysis allows scientist to better understand the climate evolution and changes.
  Eventually, the last kind of monitoring concerns the measurement of water vapor concentration and of rain fall distribution, that allows scientists to predict catastrophic floods.

- *Energy*: it embraces all missions targeting the measurement of the Earth temperature. Scientists use these measurements to monitor the evolution of global warming. The main targets of these missions are land and sea temperature
  However, this area includes also mission measuring the Albedo (i.e., relative amount of light that a surface reflects compared to the total sunlight that falls on it) and the Earth reflectance amount, two factor strongly influencing Earth temperature.

- *Land*: it integrates all studies associated with the analysis and evolution of the Earth surface from the morphological point of view. The most important application of this area is *Land topography*. Land tomography allows us to make maps of the Earth surface features. Topographic maps show the location, height, and shape of features like mountains and valleys, rivers, even the craters on volcanoes. Moreover, the applications grouped in this area have also land monitoring purposes. Some examples of these applications are active fire localization, the measurement of snow cover and snow water equivalent (i.e., the millimetres of water you would get if you converted all of that snow to liquid water), and vegetation index.

- *Life*: it covers all the applications associated with life. For example, with land cover classification scientists use satellites to make maps of many different things on Earth to help them understand how our world and life are changing. Judging by what the land surface looks like, scientists organized Earth into different categories (e.g., cities, farm lands, forests, deserts, etc.). Another important application is chlorophyll concentration measurement, that basically measures floating plants live in the ocean. These plants, called *phytoplankton*, are an important part of the ocean's food chain because many animals (such as small fish and whales) feed on them. Scientists can learn a lot about the ocean by observing where and when phytoplankton grow in large numbers.

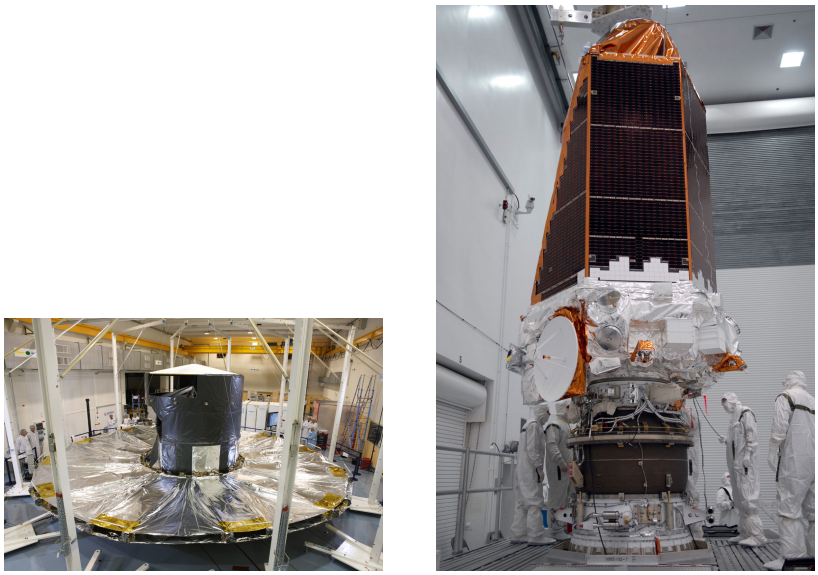
- *Ocean*: it embraces all applications concerning oceans monitoring. The most important monitoring activity performed on oceans are: *Sea ice concentration and snow extent*, that observes changes in the floating ice on the sea to predict oceans level growth, *Bathymetry*, that continuously performs an elevation map of the oceans floor to monitor its evolution along years, and *Sea surface salinity* measurement, to study the water cycle and ocean circulation, both of which are important for Earth climate analysis.

2.2 Space cartography

Scientist effort is continuously focused on the *Space cartography*, so the study of the Milky way and of the galaxy in general. This study is particularly important because allows to understand the origin of celestial body and of our planet as well, but also to discover new planets composing our galaxy.

Originally, this study was completely performed on the Earth exploiting observatories equipped with powerful telescopes. However, in the last year the need of increasing the precision in the star cataloguing and the ability to discover new planets, moved telescopes from Earth to the space. For this reason, new satellites and spacecraft have been launched to observe space environment. The last two missions targeting space cartography are NASA *Kepler* [155] and ESA *Gaia* [4] (Figure 2.8).

Kepler mission is specifically designed to survey our region of the Milky Way galaxy to discover hundreds of Earth-size and smaller planets in or near the habitable zone and determine the fraction of the hundreds of billions of stars in our galaxy that might have such planets. In order to accomplish this task, the *Kepler* satellite is equipped with a photometric instrument that is basically a *Schmidt* telescope [185]. This instrument is composed of an array of 42 *Charge Couple Devices* (CCDs), each one with 2,200 x 1,024 pixels [154]. The unique DIP operation performed on board aims at reducing the size of images to be stored. Basically, just the pixels where there are stars brighter that a threshold are stored. Each month the stored pictures are sent to Earth to be analysed.



(a) ESA *Gaia* mission (b) NASA *Kepler* mission

Figure 2.8: Space cartography missions

Instead, the *Gaia* mission aims at creating a 3D map of the *Milky Way* and at helping us to understand the origin and the evolution of our galaxy. This satellite has been launch on 19th December 2014, and is planned to spend 5 years in orbit. During these years, *Gaia* rotates on its axis while orbiting around the Sun. Thanks to two telescopes focusing light on a single billion-pixel digital sensor, it is possible to create a map of stars at a very high resolution. In particular, this digital sensor is composed of 106 CCDs, each consisting of more than eight million pixels, making *Gaia* camera boasts almost one billion pixels, so the largest focal plane ever flown in space [6].

Gaia mission makes a higher use of DIP w.r.t. the *Kepler* one. In fact, since the complete set of acquired images is huge and cannot be sent to Earth, the on-board image processing system is able to automatically process these images and extract a lot of astronomic data (e.g., star position) and star properties (e.g., age and temperature) at a very high precision.

The extremely sophisticated *Gaia* instruments will allow to create a one billion star catalogue, in which the precision of the star location is 200 times higher than the one contained in the currently used star catalogue created by the ESA *Hipparcos* satellite [5].

2.3 Satellite attitude control

In order to guarantee a proper functioning of a satellite, the control of its position and orientation in the space (i.e., its attitude) is fundamental. For this reason the *Attitude & Orbit Control System*

(AOCS) (i.e., the system aiming at continuously controlling and correcting the satellite attitude) [8] is a key element for the success of satellite operations.
An AOCS is composed by a plenty of sensors, like *Star tracker*, *Magnometer*, *Gyroscope*, and actuators, like *Trusters*, *Spin stabilization*, and *Momentum wheel*. In particular, a *Star tracker*, one of the most important sensors for absolute attitude definition, makes an extensive use of DIP.

A *Star tracker* is composed of a camera, that takes pictures to stars surrounding the satellite, and an on-board image processing system. This system analyses stars contained in the acquire image by characterizing all the possible triangular patterns that can be defined among them. Finally, the absolute attitude is defined by comparing the characterized patterns with the ones contained in the star catalogue stored on-board. For more detailed information about the pattern matching operation, the reader may refer to [121]. This method ensures a high accurate attitude control involving simple image processing operations.

2.4 Rover Navigation

A rover involved in a space exploration mission is commonly equipped with two sets of cameras: *scientific cameras*, used to take images useful to scientists to study the explored planet, and *engineering cameras*, used to automatically and/or manually control the rover navigation on the planet surface.

Engineering cameras are stereo-vision systems (i.e., a system emulating the human vision system to perform a 3D reconstruction starting from the digital images acquired from a couple of digital cameras) taking gray-scale images. In the past, these stereo-vision images have been sent to the Earth, analysed and used to plan the rover path to perform safe movement on the planet surface. Instead, in the last NASA Exploration mission, i.e., the *Curiosity rover* mission [150], the navigation system has been equipped with a high computational capability enabling the autonomous movement of the rover.

The *Curiosity rover* navigation system is composed of two main vision equipments: (i) *Navigation Camera* (Navcam) and (ii) *Hazard avoidance camera* (Hazcam) [151].
The former is a camera ensuring unmanned navigation without interfering with scientific instruments. It is composed of two pair of stereoscopic black-and-white systems, in which each camera has a 45 degree field of view and uses visible light to capture stereoscopic 3D imagery. The acquired images (Figure 2.9) are on-board processed to reconstruct the 3D representation of the planet surface in front of the rover.
The latter is made up of a set of photographic cameras mounted on the front and back of the rover. Each camera, sensible to visible light, returns black-and-white images with a resolution of 1,024 x 1,024 pixels. Images acquired by these cameras (Figure 2.10) are automatically processed on-board to map all potential hazards on the planet surface surrounding the rover.

Figure 2.9: *Curiosity* Navcam image example

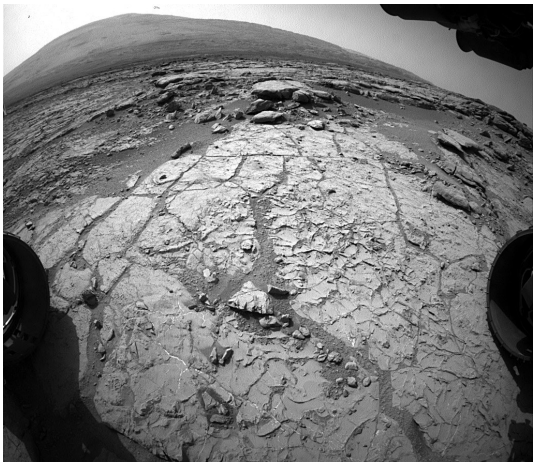


Figure 2.10: *Curiosity* Hazcam image example

Combining the information provided by *Navcam* and *Hazcam*, on 27th August 2014, *Curiosity* became the first rover able to perform an autonomous navigation on the Mars surface. In particular, *Curiosity* has been able to autonomous drive of about 43 meters in one day of navigation. This allows the reader to understand the extreme complexity of the DIP operations performed by this system.

2.5 Video-Based Navigation (VBN) landing system

During a space exploration mission, one of the most dangerous and tricky phase is the landing of the rover, or spacecraft in general, on the target planet. The landing system must guarantee a soft and safe land on the planet surface to avoid damages to the on-board instruments, that could lead to a mission failure. For this reason, engineers put a lot of efforts in the research of innovative landing systems able to both reduce to the minimum both the probability of mission failures, and the landing ellipse (Chapter 1).

The current state-of-the-art landing system is the one developed by NASA and used during the *Curiosity* mission [150]. Since this mission targets the exploration of the Mars surface, a planet protected by an atmosphere like the Earth, the instruments guaranteeing a safe land on the planet do not operate just on the proximity to the planet surface, but already when the spacecraft is approaching the entrance in the Mars atmosphere. For this reason, as described in Chapter 1, these instruments are grouped in the so called *Entry Descent and Landing* (EDL) system.

The Curiosity EDL system is mounted on the *Curiosity spacecraft* (Figure 2.11).

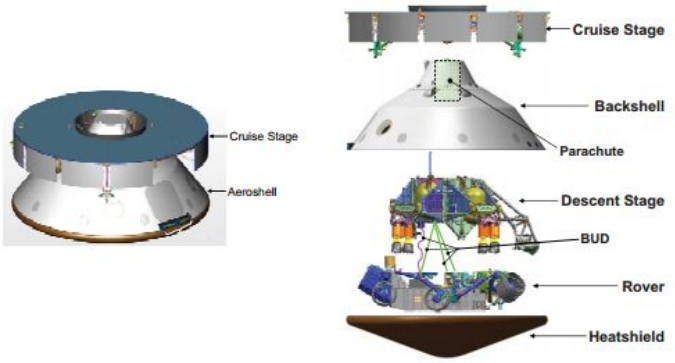


Figure 2.11: *Curiosity spacecraft* [150]

This spacecraft is composed of two main elements: *Cruise stage* and *Aeroshell*. The former aims at carrying the *Aeroshell* in the proper position on the top of Mars atmosphere, while the latter at carrying and protecting the *Curiosity* rover during the EDL phase.

In the sequel, a detailed description of the operations performed by *Curiosity EDL system* is reported [172].

When the spacecraft reaches the proper position on the top of the Mars atmosphere, the separation between *Cruise stage* and *Aeroshell* takes place (Final approach phase in Figure 2.12), and the EDL system starts its operations.

In the past, spacecrafts point the tip of the cone-shaped heat shields of the spacecraft (i.e., nose) in the direction of their motion; then they spun around their symmetry axes, and their paths were controlled by the whim of Mars' atmosphere. Differently, *Curiosity Aeroshell* performs a non-ballistic atmospheric entry. It despins, slowing from 2 revolutions per minute to halt at a specific orientation. Then, it rotates to point the tip of its nose in the direction of its travel. Just before entry, it releases two 75-kilogram tungsten masses, called *Cruise Balance Masses* (CBMs), from one side of the capsule. This operation tips the nose downward by about 20 degrees.
At this point, as shown in Figure 2.13, the *Aeroshell* faces the Mars atmosphere border and enters in it.

In the Entry phase, the *Aeroshell* just proceeds in the direction set after the CMBs release.

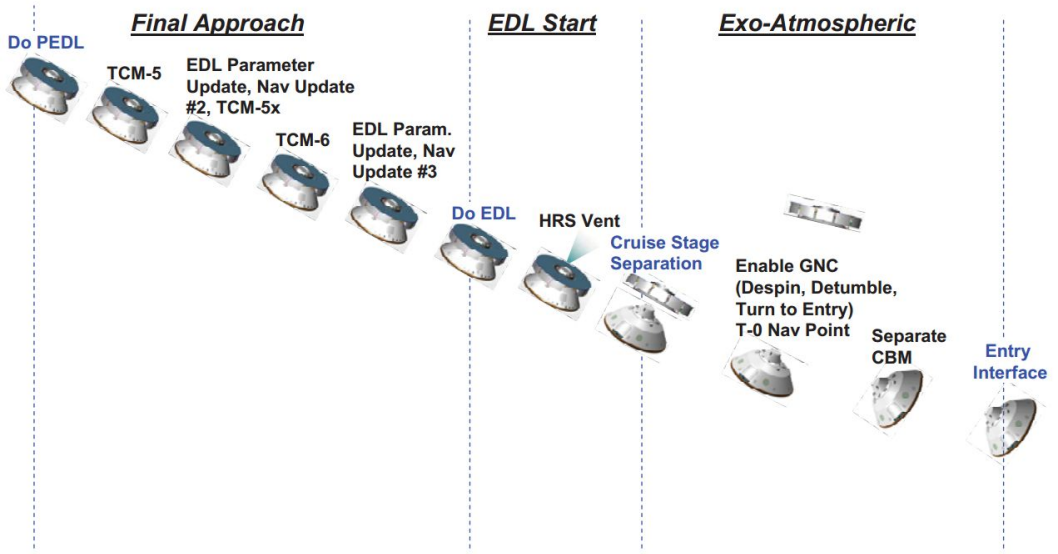


Figure 2.12: *Curiosity* EDL phase - Step 1 [150]

In this phase, all on-board instruments are protected against the high temperature due to the friction with the atmosphere border (approximately 2,100 degrees Celsius) by an heat shield.

During the parachute release, two additional manoeuvres are done.
The first, called *Straighten Up and Fly Right* (SUFR) manoeuvre, is done by releasing six 25 kg *Ejectable Mass Balance Devices* (EBDMs) to re-center the center of mass to pre-entry conditions. The second is done by performing a 180 degrees azimut turn to re-align the nose of the *Aeroshell* to the ground surface.
When the parachute is completely deployed, the heat shield is released. From this point the *Terminal Descent Sensor* (TDS) radar is activated. This instrument is a *pulsed-Doppler system* [180] that enables to accurately estimate the speed and attitude of the spacecraft during the descending phase by exploiting six independent radar beams. These information are used by the *Guidance Navigation and Control* (GNC) system to correct the descending trajectory and follow the target one. The correction of the descending trajectory is done using 8 thrusters, called *Mars Lander Engines* (MLEs), mounted on the borders of the *Descend stage* (Figure 2.11).
When the parachute slowed the vertical descending speed to 100 m/s, the *Descent stage* is separated from the *Backshell*, and it continues the guided descend trajectory.

Eventually, at an altitude of 7.5 m the rover is safely landed on the Mars surface by the *Sky Crane* system. This system allows to pose the rover on the ground without requiring airbags to protect the rover integrity.

The described *Curiosity* EDL system enabled to reduce the landing ellipse to 20 km. This high-precision delivery opened up more areas of Mars for exploration and potentially allowed

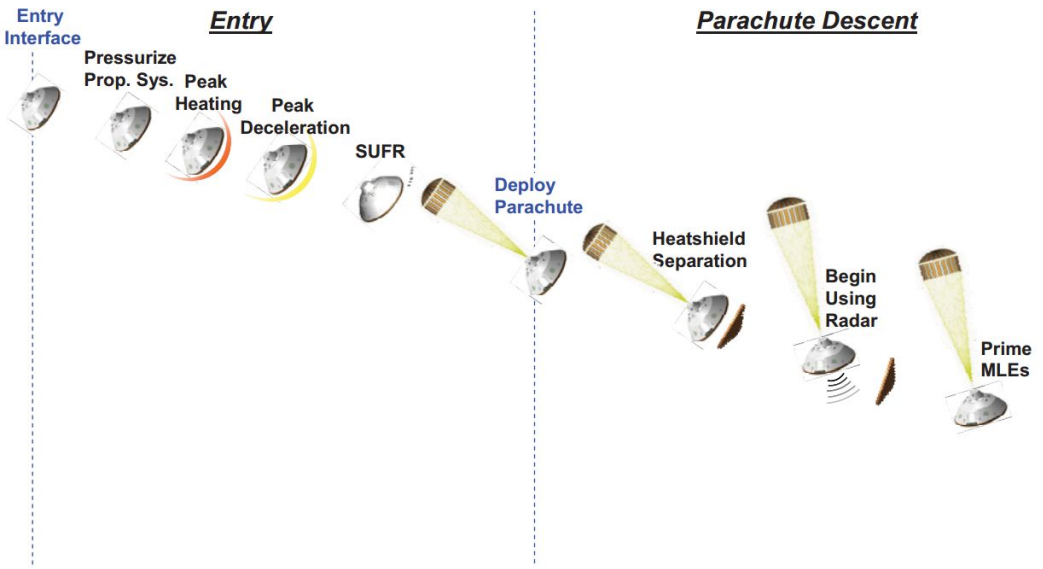


Figure 2.13: *Curiosity* EDL phase - Step 2 [150]

scientists to roam where they have not been able to before.
However, this benefit in landing accuracy came with an increased overall spacecraft mass. For this reason an emerging and promising idea is to substitute the TDS radar with a VBN system. A VBN system is able to compute the speed and attitude of a moving object by simply processing a stream of acquired images. Thus, this solution could allow to replace the heavy *pulsed-Doppler system* with a camera and a DIP board.

More in detail, a VBN system extracts geometrical information from a set of real-time sampled image frames. A video-based system can be constructed exploiting two different approaches: *Relative Navigation* and *Absolute Navigation*. The former provides the relative speed and position of the spacecraft by comparing consecutive images acquired during the descending phase. The latter compares taken images with an internal atlas of the planet surface in order to define the absolute position of the spacecraft with respect to the atlas reference system.

Differently from the Absolute Navigation, the Relative Navigation requires to know the absolute speed and position of the spacecraft at the beginning of the autonomous navigation to provide absolute measures in output. However, this is not a critical limitation since in every space exploration mission the lander (i.e., the module that practically reaches the planet surface) is released by the carried module whose position is well know during the separation from the lander.

Instead the main limitation of the Absolute Navigation is the requirement of an atlas of the target planet. In fact, this approach is only suitable for planets already visited or analyzed (i.e., the atlas has been constructed during a previous mission). For this limited applicability, the space

agencies decided to reduce the research effort in this approach to focus the research activity on relative navigation systems that potentially allow to autonomously navigate also on planet never visited before (i.e., no preliminary information on the planet to be visited are required).

In the next sections a deep analysis of the approaches and algorithms used for the relative and absolute navigation is provided. In particular, in these sections a big effort has been spent to highlight the pros and cons of different approaches, and the complexity of the exploited algorithms.

2.5.1 Relative Navigation

Relative Navigation is composed of two activities, named *Feature Extraction and Matching* (FEM), and *Motion Estimation* (ME), respectively. During FEM, each frame is processed to detect those pixels that represent *features* of interest for the image (e.g., corners or edges of surfaces). The detected features are then compared to extract those that can be recognized in two consecutive images (*matching points*). Eventually, the ME algorithms analyze the detected matching points and estimate the relative position and orientation of the camera (fixed with respect to the moving object). To increase accuracy, ME algorithms require that matching points be very accurately distributed across the entire frame [127]. While ME algorithms are not computationally intensive, FEM algorithms require high computation capability to guarantee high frame rates and therefore high accuracy.
The following sections report a detailed analysis of the most important feature extractors and matchers used for VBN.

2.5.1.1 Feature extractors

Feature extraction is the most complex activity performed by FEM algorithms. It is a special form of *dimensionality reduction*, since it enables the reduction of the number of variable to be considered. In fact, this process reduces the overall set of pixels composing the input image to a vector containing a set of information (e.g., pixel coordinate, neighbourhood pixels descriptor, strenghtness of the features, etc.) for each pixel representing a feature.

The most used feature extractors for VBN are the *Low-level* ones [64]. This kind of extractors are classified depending on the parameter considered as features. The most important classes are:

- *Edge Detectors*: they identify points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. They are mainly used in *image mosaicking*, *image registration*, *image segmentation*, and *image labelling*. The most exploited edge detectors are *Canny Edge Detector* [28], *Marr-Hildreth operator* [132], and *Differential Edge Detector* [208].

- *Corner Detectors*: they extract pixels representing the intersection between two edges. They are commonly used for *motion detection*, *image registration*, *video tracking*, *image mosaicking*, *panorama stitching*, *3D modelling* and *object recognition*. The most important and used corner detectors are *Beaudet* [18], *SUSAN* [188] and *Harris* [86].

- *Blob Detectors*: they detect regions in a digital image that differ in properties, such as brightness or color, compared to areas surrounding those regions. They are exploited in *3D modelling*, *object recognition*, *object tracking*, and *image segmentation*. The most widely used ones are *Laplacian of Gaussian* [119], *Difference of Gaussian* [216], and *Determinant of the Hessian* [122].

- *Scale-invariant feature detectors*: they identify a set of features which are invariant to image translation, scaling, and rotation, partially invariant to illumination changes and robust to local geometric distortion. These methods, differently from the ones in the previous groups, provide a descriptor for each feature, characterizing the feature and its surrounding pixels, that ease the feature matching process. These methods have the same applications of *Corner Detectors*, but they are able to provide a higher robustness. In some applications, the most important ones are *Scale-Invariant Feature Transform* (SIFT) [128] and *Speeded Up Robust Features* (SURF) [17].

Among these algorithms the most used in VBN are: *Harris corner detector*, *Scale-Invariant Feature Transform* (SIFT), and *Speeded Up Robust Features* (SURF).

2.5.1.1.1 Harris corner detector

The Harris corner detector is an interest point detector, popular due to its strong invariance to: rotation, illumination variation, and image noise [163]. Basically, this detector provides a mathematical method that allows to define if a pixel belongs to a corner or not.

As shown in Figure 2.14, considering a small windows of pixels, if the pixel in the center of the windows belongs to a corner, a small movement of the window leads to a great change in the appearance of neighbourhood pixels.

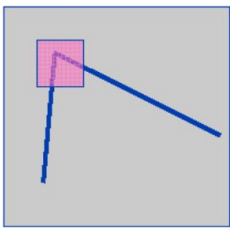
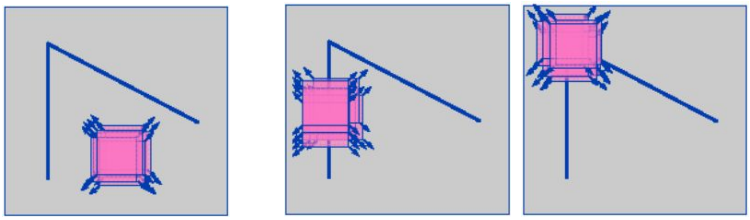


Figure 2.14: Basic idea for corners definition

In general, an image can be made up of flat region, corner or edges. The result obtained by applying the proposed idea on these three cases (Figure 2.15) can be summarized as in Table 2.1.



(a) flat region (b) edge (c) corner

Figure 2.15: Application of the Harris corner detector basic idea

Table 2.1: Result of the Harris corner detector basic idea applied to flat regions, edges and corners in a digital image

Type of image region	**Obtained result**
Flat region	No changes in the appearance of neighbourhood pixels in all directions
Edge	No changes in the appearance of neighbourhood pixels along the edge direction
Corner	Great changes in the appearance of neighbourhood pixels in all directions

Exploiting the local auto-correlation function of a signal (i.e., the mathematical operator that measures the local changes of the signal with patches shifted by a small amount in different directions), the Harris corner detector allows to mathematically define in which of these three cases a pixel is. In particular, it defines a factor called *Corner Response function* that is big iff the currently considered pixel represent a corner in the image. Equation 2.1 shows how to compute this factor.

$$R(x,y) = Det(N(x,y)) - k \cdot Tr^2(N(x,y)) \tag{2.1}$$

where *Det(X)* is the determinant of matrix *X, Tr(X)* is the trace of matrix *X, k* is a correction factor equal to 0.04, and *N(x,y)* is the second-moment matrix, defined as follow:

$$N(x,y) = \begin{pmatrix} L_x^2 & L_x L_y \\ L_x L_y & L_y^2 \end{pmatrix} \tag{2.2}$$

where L_i is the spatial image derivative in the direction *i*.

Just pixels with a high value of $R(x,y)$ can be considered valid features. Thus, to complete the feature extraction task, this parameter must be compared with a threshold. The value of

this threshold is commonly set depending on the expected image conditions in terms of noise, illumination, and contrast.

2.5.1.1.2 Scale-Invariant Feature Transform (SIFT)

Scale-Invariant Feature Transform (SIFT) has been proposed by D. Lowe in 1999. This algorithm provides strong invariant features to illumination variations, noise, scale modifications, and rotations.

SIFT is composed of four main steps: (i) *Scale space extrema detection*, (ii) *Keypoints localization*, (iii) *Orientation computation*, and (iv) *Keypoint descriptors extraction*.

Scale space extrema detection is the first operation performed on the input image. First, it requires to create the scale space associated with the input image. This task is accomplished by creating the *scale image pyramid* (Figure 2.16).

Figure 2.16: Scale image pyramid example

This pyramid is composed of different octaves (rows in Figure 2.16). Images populating an octave represent different scale level. They are obtained by blurring the first image of the octave with a *Gaussian blurring function* (Equation 2.4) with an increasing variance (σ^2). Instead, to pass from one octave to the next one the height and width of the first image composing the previous octave must be divided by two. For the sake of completeness, Equation 2.3 shows how to compute the pixel values of the first image composing the second octave.

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{2.3}$$

where $L(x,y,\sigma)$ is the pixel value of the first image in the second octave, $*$ is the convolution mathematical operator, $I(x,y)$ is the input image pixel value, and $G(x,y,\sigma)$ is the *Gaussian blurring function* defined as:

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2-y^2)}{2\sigma^2}} \tag{2.4}$$

The scale space extrema detection is completed by approximating the *Laplacian of Gaussian* (LoG) through the *Difference of Gaussian* (DoG). DoG are defined among images composing each octave and defining the pixel representing an extrema (minimum or maximum) in the created pyramid in a neighbourhood pixels window. Basically, the DoGs are computed by algebraically

subtracting pixel by pixel each consecutive couple of image composing an octave, as shown in Figure 2.17.

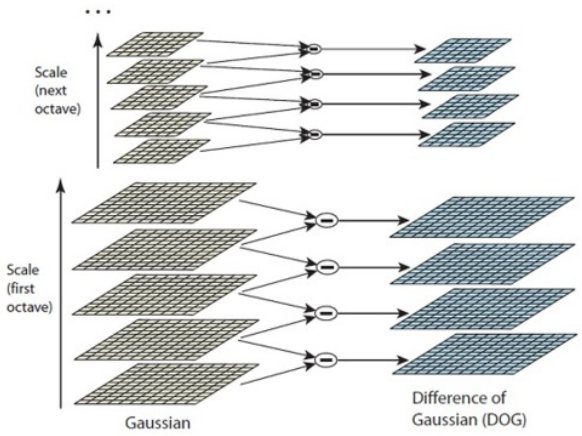


Figure 2.17: DoG computation example

The extrema search in the neighbourhood pixels window is done, as in Figure 2.18, by comparing each pixel in an image with its 8 neighbours as well as with the 9 neighbourhood pixels, in the next and previous scale (i.e., the next and previous images in the current octave). If the pixel is an extrema (i.e., a maximum or a minimum among the 26 neighbourhood pixels) it is selected as a *keypoint* and it is stored together with the associated scale.

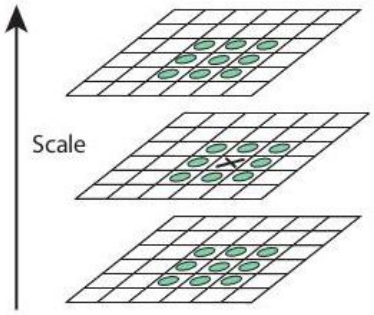


Figure 2.18: Pixel comparison for extrema definition

Keypoint localization aims at refining the defined extrema (i.e., keypoints). Basically, this operation eliminates unstable keypoints from the final list by finding those that have low contrast or are poorly localized on an edge. Low contrast keypoints are detected by simply comparing the keypoint intensity with a threshold. Instead, keypoints lying on an edge are detected by computing the *Hessian function*, associated to the currently considered pixel, and comparing its two eigenvalues. Since an edge is characterized by a large principal curvature across its direction but

a small one in the perpendicular one, and the Hessian function eigenvalues are proportional to these two principal curvatures, when the eigenvalues differ too much the keypoint lies on an edge [237].

Orientation computation aims at assigning a consistent orientation to the keypoints based on local image properties. This information added to each keypoint ensures to reach a rotation invariance. The orientation is defined through the *gradient magnitude* (m) and the *gradient orientation* (μ), that can be computed as show in Equation 2.5 and 2.6, respectively.

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{2.5}$$

$$\mu(x,y) = tan^{-1} \frac{(L(x,y+1) - L(x,y-1))}{(L(x+1,y) - L(x-1,y))} \tag{2.6}$$

Eventually, *Keypoint descriptor extraction* is done by measuring the local image gradients at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination. The local gradient data, used above, is also used to create keypoint descriptors. The gradient information is rotated to line up with the orientation of the keypoint and then weighted by a Gaussian with a variance of 1.5 multiplied by the keypoint scale. These data are then used to create a set of histograms over a window centred on the keypoint. Keypoint descriptors typically use a set of 16 histograms, aligned in a 4x4 grid each with 8 orientation bins, one for each of the main compass directions and one for each of the mid-points of these directions. This process results in a feature vector containing 128 elements.

For more detailed information about SIFT, readers may refer to [128].

2.5.1.1.3 Speeded Up Robust Features (SURF)

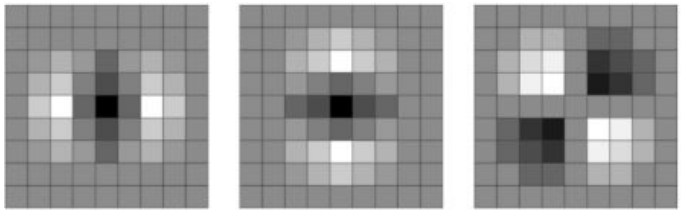
Speeded Up Robust Features (SURF) is a feature detector and descriptor proposed by Herbert Bay in 2008 [17], that provides the same robustness guaranteed by SIFT (Section 2.5.1.1.2), but increasing the timing performance. This algorithm follows the same flow executed by SIFT, but it changes the way to perform some of the required steps.

Differently from SIFT, SURF uses a *Hessian-based blob detector* to find interest points. In particular, SURF performs non-maximal-suppression of the determinants of the *Hessian matrices*. The determinant of a *Hessian matrix* expresses the extent of the response and is an expression of the local change around the area [17].

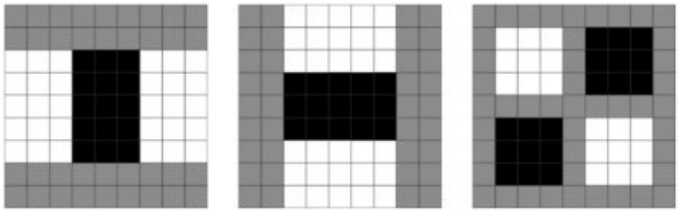
Since the Hessian matrix is composed of the image second order derivative along the *x*, *y* and *xy* directions, and the convolutions is very costly to calculate, the derivatives computation is approximated and speeded-up with the use of *Box filter* and *integral images* [17].

Box filters are approximated kernel to compute the second order derivative along the *x*, *y* and

xy directions. Figure 2.19, from left to right, shows the approximation of second order Gaussian kernels (Figure 2.19(a)) with *Box filters* (Figure 2.19(b)) for the derivative computation along the *x*, *y* ans *xy* direction.



(a) Second order Gaussian kernels



(b) Box filters

Figure 2.19: Approximation of second order Gaussian kernels with Box filters (gray areas represents zero factor, black ones -1 factors, and white ones +1 factors)

To boost up the execution speed the convolution computation with Box filters, SURF operates on integral images.
An integral image allows to speed up the sum of pixels in a rectangular region. In particular, the time cost to compute this sum is independent from the rectangular region size. To better understand this concept, it is necessary to define the process to compute an integral image. Equation 2.7 reports the general formula to compute pixel values composing an integral image.

$$II(x,y) = I(x,y) + II(x-1,y) + II(x,y-1) + II(x-1,y-1) \tag{2.7}$$

where $II(i,j)$ and $I(i,j)$ are the pixel value in the (i,j) position of the integral and input image, respectively.
If a pixel required by Equation 2.7 falls outside the image, its value must be set to zero. For a better comprehension, Figure 2.20 shows an example of integral image computation.
Eventually, the sum of pixels in a rectangular region can be computed as shown in Equation 2.8

$$\sum_{x=x_a}^{xb}\sum_{y=y_a}^{y_b} I(x,y) = II(x_b,y_b) - II(x_a-1,y_b) + II(x_b,y_a-1) + II(x_a-1,y_a-1) \tag{2.8}$$

where (x_a,y_a) and (x_b,y_b) represent the upper-left and bottom-right corners of the rectangular region, respectively.

5	2	3	4	1
1	5	4	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

(a) Input image

5	7	10	14	15
6	13	20	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

(b) Integral image

Figure 2.20: Integral image computation example

Thus, independently from the rectangular region size, the pixel sum can be computed with just four add operations. As in the integral image computation, if one of the pixel values required by Equation 2.8 falls outside the image, its value must be set to zero. For the sake of completeness, Figure 2.21 shows two examples of pixel sum computation.

5	**2**	**3**	4	1
1	**5**	**4**	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

5	7	10	14	15
6	13	**20**	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

5 + 2 + 3 + 1 + 5 + 4 = 20 - 0 - 0 - 0 = 20

(a) Rectangular region on the image border

5	2	3	4	1
1	**5**	**4**	**2**	3
2	**2**	**1**	**3**	4
3	5	6	4	5
4	1	3	2	6

5	7	10	**14**	15
6	13	20	26	30
8	17	25	**34**	42
11	25	39	52	65
15	30	47	62	81

5 + 4 + 2 + 2 + 1 + 3 = 34 - 14 - 8 + 5 = 17

(b) Rectangular region on in the middle of the image

Figure 2.21: Pixel sum computation example

Another important benefit provided by the combination of Box filters and integral image concerns the constant computational cost of the convolution with increasing kernel size. For this reason, to create the scale space in a time efficient way, in SURF, instead of scaling the picture size while maintaining the size of the kernel constant (as in SIFT), the size of the image is maintained constant among octaves while the kernel size is increased (Figure 2.22).

Another difference between SURF and SIFT concerns the way to compute feature descriptors.

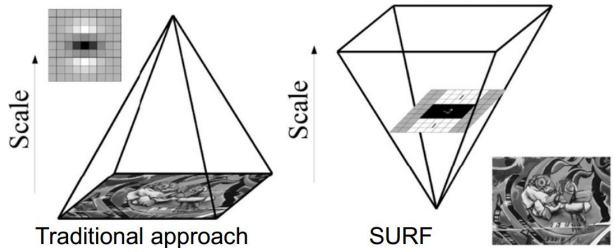
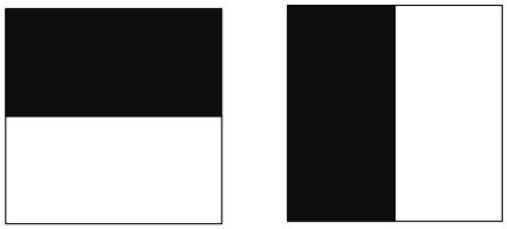


Figure 2.22: Scale space creation with the traditional and the SURF approach

Instead of using gradients as in SIFT, SURF exploits *Haar wavelet responses* along the x and y directions [17]. Basically, around each feature is defined an interest area as big as the feature scale multiplied by 20. The interest area is divided into 16 subareas that are described by the values of a wavelet response in the x and y directions. These components are computed as a convolution between *Haar kernels* (Figure 2.23) and the interest area, exploiting the speed-up provided by integral images and Box filters.



(a) Wavelet component along y

(b) Wavelet component along x

Figure 2.23: *Haar kernels* for wavelet components computation (black areas represent a -1 factor, and white ones a +1 factor)

The interest area are weighted with a Gaussian function centred on the interest point to give some robustness against deformations and translations. For each subarea a vector is defined, based on 5x5 samples, as shown in Equation 2.9.

$$v = \left\{\sum \delta x, \sum \delta y, \sum |\delta x|, \sum |\delta y|\right\}, \tag{2.9}$$

where δx and δy are the *Haar wavelet responses* along x and y direction, respectively.

Eventually, the sixteen vectors are concatenated to create the feature descriptor. Since each vector is 4 components long, a SURF feature descriptor contains 64 elements.

For more detailed information about SURF, the reader may refer to the original paper published in [17].

2.5.1.2 Feature matchers

In VBN applications, feature matching is a fundamental step since it ensure to create the correspondence among features in two consecutive images.

Feature matchers for VBN applications can be grouped into two main categories [2]:

- *Area-based matchers*
- *Features-based matchers.*

The following sections provide a deep analysis of these two categories, highlighting the main characteristics and involved algorithms.

2.5.1.2.1 Area-based matchers

Area based matching methods determine the correspondence between two image areas (*reference windows*) according to the similarity of their pixel values.

The main issue for applying this kind of matchers concerns the definition of the reference windows location. Commonly, to solve this issue, reference windows are centred on the extracted image features, and the difference between the two windows is computed to define whether the features match or not.

The similarities between the two image areas can be quantified exploiting many algorithms. The most used ones are *Sum of Absolute Difference* (SAD), *Sum of Square Difference* (SSD), *Cross-Correlation* (CC), and *Normalized Cross-Correlation* (NCC).

Sum of Absolute Difference (SAD) is the simplest approach. As shown in Equation 2.10, it just sums the absolute differences between the two image templates.

$$SAD = \sum_{i=0}^{M} \sum_{j=0}^{N} |I_1(x_{f1} + i, y_{f1} + j) - I_2(x_{f2} + i, y_{f2} + j)| \tag{2.10}$$

where M and N are respectively the height and width of the reference windows, I_1 and I_2 are the two consecutive images, and (x_{f1}, y_{f1}) and (x_{f2}, y_{f2}) are the feature coordinates in the first and second image.

Sum of Square Difference (SSD) sums the square differences between the two image templates. Equation 2.11 reports the formula to compute SSD.

$$SSD = \sum_{i=0}^{M} \sum_{j=0}^{N} (I_1(x_{f1} + i, y_{f1} + j) - I_2(x_{f2} + i, y_{f2} + j))^2 \tag{2.11}$$

Adding the square differences, instead of the absolute ones, allows to strongly discriminate the disparity between the two image reference windows.
Since the SAD and the SSD formulas quantify the distance between the two image reference windows, just when the result of these computations is lower than a certain value the two features can be considered as a valid match.

Differently from SAD and SSD, the two correlation methods (CC and NCC) provide similarity measurement. Thus, instead of quantifying the differences between the reference windows, CC and NCC provide high values when the considered image template are close to each other. Thus, just when this value is higher than a threshold the computed match can be considered as valid. Equations 2.12 and 2.13 are used to compute CC and NCC, respectively.

$$CC = \sum_{i=0}^{M} \sum_{j=0}^{N} I_1(x_{f1} + i, y_{f1} + j) * I_2(x_{f2} + i, y_{f2} + j) \tag{2.12}$$

$$NCC = \sum_{i=0}^{M} \sum_{j=0}^{N} \frac{I_1(x_{f1} + i, y_{f1} + j) * I_2(x_{f2} + i, y_{f2} + j)}{\sqrt{\sum_{i=0}^{M} \sum_{j=0}^{N} (I_1(x_{f1} + i, y_{f1} + j))^2 * \sum_{i=0}^{M} \sum_{j=0}^{N} (I_2(x_{f2} + i, y_{f2} + j))^2}} \tag{2.13}$$

Comparing the complexity of these two formulas, it can be immediately noted that NCC is characterized by a higher complexity, while CC requires a similar computational cost of SAD and SSD.

From the accuracy point of view, correlation methods provide high performances than SAD and SSD [240] [74]. In particular, NCC is even more robust than CC, since, normalizing the correlation value with the values contained in the two reference windows, is practically insensible to illumination condition variations between the two consecutive images.
Eventually, since these methods use a fixed reference windows are not able to provide accurate results when high rotations or scale variations are present between the two consecutive images.

2.5.1.2.2 Features-based matchers

Differently from *Area-based* approaches, *Features-based matchers* require feature descriptors as input (Sections 2.5.1.1.2 and 2.5.1.1.3). Thus, they cannot be used with features extractors that provide in output just feature locations (e.g., Harris corner detector in Section 2.5.1.1.1).
Basically, these methods search among the set of feature descriptors, extracted from two consecutive images, the most similar one.
Feature descriptors matching can be done in different ways. In the sequel the most important methods are reported .

The simplest one quantify the distance by summing the differences among components of two feature descriptors. Commonly, this task is done exploiting the SSD (Equation 2.11. Another common approach is the computation of the euclidean distance between features descriptors. In both cases, the computation must be performed for all the feature descriptors of the two consecutive images, and just the couples having the distance lower than a certain value can be considered valid matches. For this reason these methods, also called *brute force approaches*, are really timing consuming especially when the feature descriptors dataset is big.

In order to reduce the computation time, advanced methods, that make use of special kind of data structure allowing to split the feature descriptors dataset into smaller hyper-boxes, have been invited. The most efficient methods makes use of *k-d trees* [186] as data structure.

In general, differently from the *Area-based matchers*, *Features-based matchers*, when work on descriptors that are scale and rotation invariant (e.g., the ones extracted with robust extractors like SIFT (section 2.5.1.1.2) and SURF 2.5.1.1.3)), are efficient even in presence of strong rotation and scale modification between the two consecutive images. However, they are more time consuming since the number of elements composing a feature descriptor is higher than the number of pixels composing an image reference window used in *Area-based matchers*. Thus, the number of operations to verify if a feature couple is a valid match is higher.

2.5.2 Absolute Navigation

As opposed to relative navigation, which depends on a unique origin or previous motion information to locate the current position of a space craft, absolute navigation computes the location of a space craft without a unique origin point or previous motion information using only the available terrain information. In other words, it extracts the spacecraft position referred to a general, fixed, coordinate system.

The basic approach consists in extracting some kind of information, from the images taken by the spacecraft, characterizing the landing zone and trying to match these information with the ones stored in a reference database.
In order to ensure the proper definition of the absolute position, the database stores the information related to a planetary surface portion wider than the target landing area.

Existing approaches for Video-Based Absolute Navigation can be classified into three main groups:

- *Orbiter-assisted*
- *Shape-based*
- *Crater-based.*

The *Orbiter-assisted* approach [169] is depicted in Figure 2.24. It requires the presence of an orbiter, or satellite, equipped with a high resolution camera. As shown in Figure 2.24, the Field of View (FoV) of the spacecraft camera must be completely covered by the FoV of the orbiter camera.
The picture taken by the Orbiter has the purpose of reference image. The reference database is created by extracting features from this image. The image taken by the spacecraft is firstly scaled and rectified according to the orbiter attitude, provided by the *Inertial Measurement Unit* (IMU) (i.e., a module equipped in every spacecraft involved in a space exploration mission), in order to align the picture with the one taken by the orbiter (i.e., after rectification the two pictures lie on the same plane). After that, features are extracted from the rectified picture, and compared with the database. After features matching, since the orbiter position is known, the absolute position

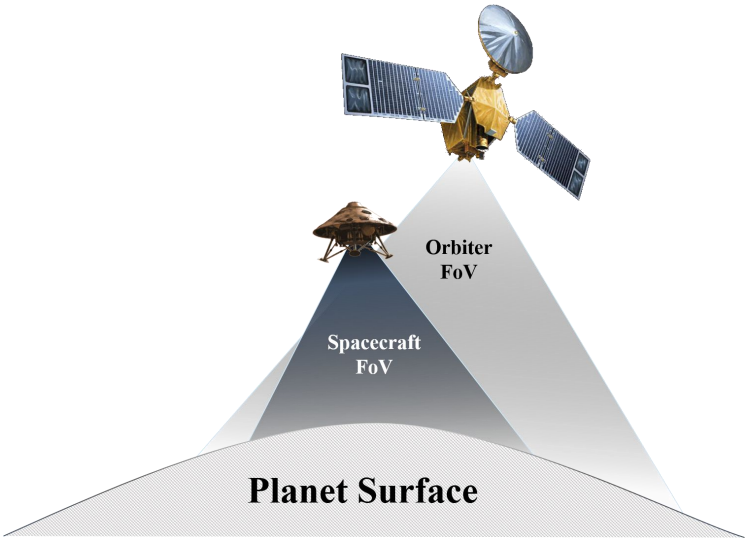


Figure 2.24: Orbiter-assisted Absolute Navigation

of the spacecraft can be defined with respect to the orbiter reference system.
This method has a lot of benefits, since it provides a good precision, it does not require specific landmark on the planet surface, it does not require an internal database, and it is not susceptible to different illumination conditions, since the compared pictures are acquired at the same time. However, the method is based on two main hypothesis that are not feasible for the majority of space exploration missions. In particular, it requires:

- the presence of an orbiter that monitors the overall descending phase of the spacecraft;
- a permanent communication channel between the two entities, that is not always guaranteed during the spacecraft descending phase.

The *Shape-based* and *Crater-based* [33] approaches are both based on a pre-computed reference database.
In the former, the information composing the database are landmarks templates, characterized by an absolute position w.r.t. a reference system. The image rectification is performed to align the acquired pictures with the one composing the surface atlas (in the sequel called atlas image). This task is accomplished by exploiting the current attitude and altitude of the descent module. After image alignment, regions of interest (i.e., landmarks) are extracted from the spacecraft picture, and they are compared with the reference database to find the most robust match. Commonly, the matching task is done by extracting from the spacecraft acquired images a set of image template containing region of interest. These templates are then stepped over the atlas image to find the regions having the higher number of pixels in common, and so to find a valid matches. Even-

tually, after the matching task, from the absolute position characterizing the matched database template, it is possible to define the absolute position of the spacecraft.
As it can be noted from this brief description, *Shape-based* methods involves simple operations and work for different kind of landmarks. However, they are not able to provide good precision when the planet surface contains landmarks with similar shape, and, in addition, they are not invariant to the spacecraft attitude.

The *crater-based* approach exploits a similar method but it exclusively uses craters as landmarks. The crater matching can be commonly done in two different ways: *Conic-pair Invariants matching* [118] or *Cross-correlation matching* (Section 2.5.1.2).
In the former, for each possible pair of craters in the acquired pictures and in the atlas image, an invariant operator, called *conic-pair invariant*, is defined. This operator allows to create a signature, invariant to geometrical transformation, identifying each couple of craters. Thanks to the invariance of this operator, the matching can be done without knowing the attitude of the spacecraft in the atlas reference system. Thus, the spacecraft image rectification can be avoided. However, this methods presents an important drawback, being efficient only if the planet surface is picked by a high number of craters.
Instead, in the cross-correlation matching, for each crater detected in the acquired image, a surrounding image template is extracted. This template is then stepped over the atlas image in order to find a crater match through the cross-correlation operation (Section 2.5.1.2). The main positive aspect of this approach concerns the required simple operations that allow to limit the execution time. However, this methods is not able to provide acceptable performance when the atlas image and the acquired images have different photometric properties (i.e., illumination, contrast, etc.). Moreover, this method is not invariant to the spacecraft attitude, it can find a lot of fake matches when different craters have similar structure, and it requires a huge dataset to store the atlas image.

Even if *Shape-based* and *Crater-based* approaches have some drawbacks, they are preferred to the *Orbiter-assisted* ones since they present less critical limitations.
Analyzing the literature, a lot of these methods have been presented.

[146] proposes a VISion aided Inertial NAVigation (VISINAV) system. It is an hybrid relative and absolute navigation system for planetary landing applications. It utilizes both information extracted from an IMU and pictures acquired from an imaging camera. The visual measurements are combined in an optimal fashion with measurements from an IMU, to produce estimates of the spacecraft position and attitude during EDL.
As aforementioned, this system does not implement a pure absolute navigation approach, since it combines the extraction and matching of two types of features: *mapped landmarks* (i.e., craters whose global coordinates can be inferred from a map) and *opportunistic features*. Just the former are related to the absolute navigation, while the latter represent the features used in the standard relative navigation computation described in section 2.5.1.1.

Taking into account just the absolute navigation part of VISINAV, the landmarks matching is done with a cross-correlation based crater matching, optimized to reduce the required computation time. In order to avoid to step the crater templates over the overall atlas image, a Fast Fourier Transform (FFT) map matching forego the actual landmarks matching. This added task allows to rapidly approximate the horizontal position of the spacecraft w.r.t. the atlas reference system. This approximation, even if not really accurate, enables to create a searching window inside the atlas image that strongly reduces the searching area of potential landmark matches.
However, this approach, even if it provides a really high execution speed, it embeds all the drawbacks of the cross-correlation based crater matching.
In [32] an approach based on the *Conic-pair Invariants matching* is proposed. This work increases the robustness of the matching task by comparing, in addition to the *conic-pair invariant* associated with each crater couples, a set of peculiar crater attributes. In particular, to remove potential fake matches identified by the basic method, this work compares also the radius and the orientation of each detected craters. This improvement allows to increase the robustness of the *Conic-pair Invariants matching*, but, at the same time, it increases the size of the reference database, resulting in a higher memory requirement.

[34], in order to even more increase the matching robustness, exploits in parallel the *Conic-pair Invariants matching* and *Cross-correlation matching*, plus the so called *context based matching* [57]. This considers the positions and sizes of a set of craters found in an image. Their constellation usually forms a unique pattern, which could be used to identify the correspondences between the acquired images during the descending phase and the atlas image. Basically, this approach randomly selects three craters in the acquired image. These three craters have to form a triangle in order to avoid an ill conditioned problem. Knowing the attitude and altitude of the spacecraft (provided by the IMU), the epipolar geometry (Section 2.6.1) can be generated and used, together with the size and shape of each crater to find the corresponding crater triplets in the atlas and spacecraft images.
The combination of these techniques provides a great increase in the accuracy, but at the same time, it strongly increase the execution time. In fact, the main drawback of this techniques is the huge time required to find valid matches, that is not compatible with the timing performance required by an absolute navigation system. In addition, the memory requirements are higher than the previously presented methods, since the internal atlas must contain the information required by all of the three exploited matching approaches.

Eventually, [195] proposes a hardware infrastructure that implements an innovative crater-based absolute navigation . This solution aims at identifying each landmark (i.e., craters) through a characterizing point, and then applying the same approach used in Planar triangle-based (PT) Star trackers [36]. PT Star trackers are used to define the attitude of a satellite depending on which stars are within the satellite camera field of view. The basic steps performed by a PT Star tracker are: (i) develop a triangle from a combination of three stars, (ii) characterize the triangle with

its area and polar moment, and, eventually, (iii) compares the extracted area and polar moment with the database of all possible star triangles in order to find a match.
In this approach the characterizing points associated with landmarks act as stars.
The proposed method can be split in two main tasks: *Reference database development*, to be performed off-line, and *Absolute position estimation*, executed at run-time. The database development task is carried out by creating a surface map of the landing zone. The surface map can be obtained by stitching a set of images taken by satellites orbiting around the target planet. Then, landmarks, i.e., craters, are extracted and the associated characterizing points are defined. Characterizing points are defined in terms of centroids of the detected craters, and the associated absolute position. In addition to centroids, the database contains the set of all possible triangles (among all centroids), defined by the triangle surface area, perimeter and polar moment.
This approach strongly reduces the database size, w.r.t. existing methods. In this approach the database requires three numeric values for each triangle, plus the absolute position of each centroid, instead of an image template for each landmark.
The *Absolute position estimation* is a sequence of consecutive operations to be performed on the input image at run-time. First, the images acquired by the spacecraft, during the descending phase, are pre-processed to reduce the level of noise and enhance the overall quality. Image pre-processing mainly consists of *noise filtering* (e.g., by applying a Gaussian filter [81]) and *rectification*, needed to align the plane of the acquired spacecraft image with the plane of the images used for the database development. The rectification process is essential in order to apply the Planar Triangle-based Star tracking algorithm, that in general, does not take into account the spacecraft attitude because stars can be considered as point at an infinite distance. On the other hand, during the descending phase, craters, and consequently the associated centroids, changes their relative position in the camera field of view depending on the spacecraft attitude.
These pre-processing operations ensure to increase the performances of the following image processing algorithms.
After image pre-processing, craters are extracted. This operation can be accomplished by using image segmentation [191]. This kind of algorithm provides better results with respect to a standard edge detector (e.g., Canny [28]), since it does not require information about the sun elevation and it is more robust to illumination condition variations.
If less than three craters are extracted, the star tracking approach cannot be applied and a new picture must be acquired. Otherwise, each extracted crater is characterized by the centroid.
Exploiting the altitude information, provided by a LIDAR sensor (commonly equipped on spacecrafts involved in space exploration missions), the absolute distance among centroids is computed. Then, all possible triangles are defined, and characterized by the area, perimeter and polar moment. Finally, these information are compared with the ones stored in the database to find potential matches. The matching task is very simple since it consists of searching numerical values in the pre-computed database table. Matches are used to compute the absolute position

of the spacecraft by exploiting the absolute position of the centroids composing the matched triangles.
However, even if this approach solves many issues in terms of memory requirement and execution speed, it requires to land on a surface picked by at least three craters, making this approach not suitable for every landing site.

From this analysis of the current state-of-the-art, it can be noted that an efficient way to solve all the issues characterizing the absolute navigation system has not yet been found. Thus, a huge research effort must still be spent on this topic. However, as already mentioned at the beginning of this chapter, the space agencies are reducing their research efforts on this approach in favour of relative navigation approaches.

2.6 Active Space Debris removal

The challenge of removal of large space debris, such as spent launcher upper stages or satellites having reached the end of their lifetime, in low, medium, and geostationary earth orbits, is already well-known. It is recognized by the most important space agencies and industries as a necessary step to make appreciable progresses towards a cleaner and safer space environment [68, 148]. This is a mandatory condition for making future space-flight activities safe and feasible in terms of risks. Space debris, defined as non-functional objects or fragments rotating around or falling on the earth [100], are becoming a critical issue. Several studies and analysis have been funded in order to identify the most appropriated approach for their removal. Recent studies demonstrated that the capability to remove existing space debris, over preventing the creation of new ones, is necessary to invert the growing trend in the number of debris that lie in orbits around the Earth. Nowadays, the focus is on specific space debris, weighting about 2 tons and spanning about 10 meters [170]. This class of orbiting debris is the most dangerous for aircrafts and satellites, representing a threat to manned and unmanned spacecrafts, as well as a hazard on Earth, because large sized objects can reach the ground without burning up in the atmosphere. In case of collision, thousands of small fragments can potentially be created or even worst they can trigger the Kessler syndrome [112]. An example of this class of debris is the lower stage of solid rocket boosters, such as the third stage of Ariane 4, the H10 module, usually left from European Space Agency (ESA) as space orbiting debris [23, 67].

The basic procedure for removing a space debris consists of three steps. The first phase is the *debris detection and characterization*, in terms of size, shape profile, material identification, and kinematics. The second phase, called *non-collaborative rendez-vous*, exploits the information gathered from the first phase in order to identify the best approach (e.g., trajectory) to capture the identified debris. Finally, in the *capture and removal* phase, depending on the on-board functionalities of the chaser satellite, the debris is actually grappled, de-orbited from its position, and safely guided during the descending phase on the Earth atmosphere. [110, 244].

Concerning this basic solution, many different approaches to perform the aforementioned steps have been proposed (e.g., *e.Deorbit* [20], *CleanSpace One* [131], *Sling-Sat Space Sweeper* [142], and *Solar Sail* [94]).
The goal of these missions is to study sophisticated imaging sensors for an autonomous debris characterization, and advanced capture systems.
Several capture mechanisms are being studied in parallel to minimise mission risk. *Throw-nets* [221] have the advantage of scalability, a large enough net can capture anything, no matter its size and attitude. Tentacles [221], a clamping mechanism that builds on current berthing and docking mechanisms, could allow the capture of launch adapter rings of various different satellites. Harpoons [221] work no matter the target's attitude and shape, and do not require close operations. Robotic arms [95] are another option.
In addition to the basic active space debris removal approach, other potential procedures have been proposed. For example, the *Japanese Aerospace Exploration Agency* proposed [107] the usage to use an electrodynamic tether whose current would slow down the speed of satellites or space debris. Slowing the satellite speed would make it gradually fall closer to Earth, where it will burn up. Instead, *Space Debris Elimination* (SpaDE) [84] would push satellites into a lower orbit by using air bursts within the atmosphere. Debris pushed on the lower orbit will slowly decay entering in Earth atmosphere, where they will burn up.
A design proposal from Daniel Gregory of Raytheon *BBN Technologies* in Virginia [45] would use a balloon or high-altitude plain to send the bursts out.
However these two proposals have a common drawback. Since debris are also of big size, the complete burst of them due to the impact with the Earth atmosphere cannot be guaranteed. For this reason, since the descending trajectory of the decaying object cannot be predicted a priori, the space debris can hit the Earth surface even in populated areas. Thus, the basic debris removal approach is currently the preferred one.

My research activity on this topic has been mainly focused on the imaging sensors to compute the autonomous characterization of space debris. In order to collect the required information about the object to be removed, three main operations must be performed: (i) the debris three-dimensional shape reconstruction, (ii) the definition of the structure of the object to be removed and the identification of the composing material, and (iii) the computation of the kinematic model of the debris. In particular my activity focuses on the first of these three phases.

Since space applications impose several constraints on allowed equipments in terms of size, weight, and power consumption, many devices commonly used for 3D object shape reconstruction cannot be used when dealing with space debris removal (e.g., laser scanners [200] and LIDARs [111]). Moreover, the chosen device should be passive, not only for power constraints, but also because passive components are more robust against damages caused by unforeseen scattering of laser light.
The easiest (and probably cheapest) device suitable for space missions is a digital camera acquir-

ing visible wavelengths. Either based on CCD or CMOS technology, power consumption of digital cameras is affordable as well as the further processing of provided images.
Digital cameras can be used for 3D shape reconstruction exploiting several techniques. The most used and effective techniques to compute a 3D model starting from visible light images are: (i) *Stereo-vision systems* and (ii) *Shape from Shading*. Next sections report a deep analysis of these two techniques.

2.6.1 Stereo-vision system

A *stereo-vision system* mimics the human visual system with two (or more) points of view, and provides in output the so called *dense map*: an image that point out the distance from the observer of each pixel composing the input image.

This section provides a survey of the state-of-the-art procedures able to reconstruct a dense map, providing dense and full 3D reconstructions of objects from multiple views.
A common stereo-vision system is made up of two cameras, watching at the same scene, that emulate the human binocular vision. These two cameras have a fixed distance called *inter-camera distance*. The output of a stereo-vision system is a couple of images (stereo images) that provide the left and right side view of the scene in front of the system.
Figure 2.25 shows how a point in the 3D space is reproduced in the stereo images.

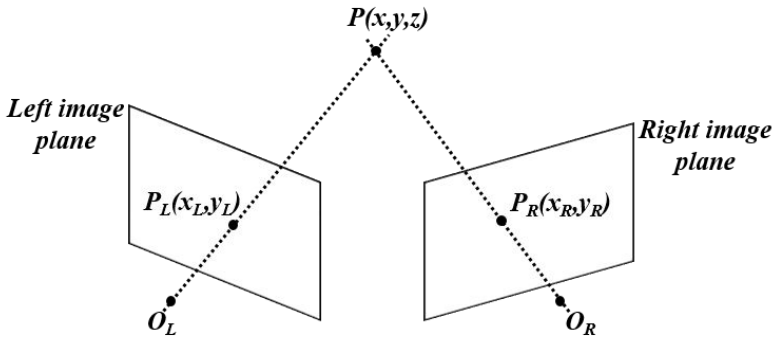


Figure 2.25: Stereo-vision system

A point $P(x, y, z)$ in 3D space is seen by the left view, which we will call the left image, as a projection point $P_L(x_L, y_L)$. This point is located where the line between the left camera focal point (O_L) and $P(x, y, z)$ intersects the image plane of the left image. The same consideration can be done for the right view, identifying the $P_R(x_R, y_R)$ projection point.
In order to translate the content of the stereo vision images in the 3D space, the pixels composing in the left view image must be matched with the one in the right side image. Then, knowing the intrinsic and extrinsic parameters characterizing the system, the locations of the matching pixel can be translate as the $P(x, y, z)$ point in 3D space.

In order to ease the matching process, *epipolar geometry* is used. Figure 2.26 shows the epipolar geometry applied on the point $P(x, y, z)$. The basic element of the epipolar geometry is the

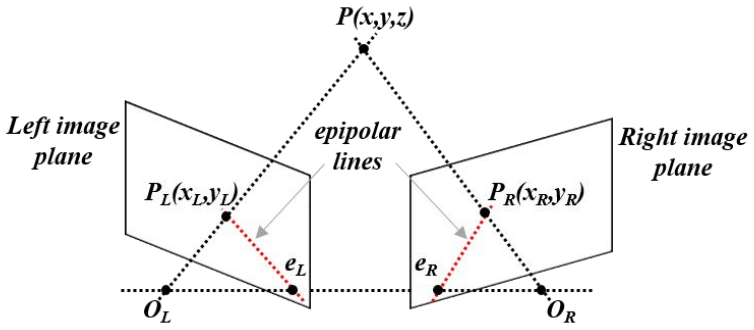


Figure 2.26: Epipolar geometry application

epipole. In a stereo vision system there is an epipole for both the left and right view. To identify the location of these points is necessary to draw a line connecting the left and right camera focal points. The line connecting each epipole to the projection point in the same image plane is called *epipolar line*. Given both the cameras intrinsic and extrinsic parameters, we can generate, starting from the epipolar line in the left image, the corresponding epipolar line in the right image. Since matching pixels can be located only on correspondent epipolar lines, the search space of matching point is reduced, from the overall image, to the pixels lying on the epipolar line.
In order to understand how to practically compute dense map of objects, in the sequel are reported the main steps to be performed. These steps could be grouped into two main categories or "phases of execution": (i) *calibration*, one shot phase, and (ii) *effective dense map computing*, real-time phase.

2.6.1.1 Calibration

As aforementioned, the matching process based on the epipolar geometry requires, for each pixel in the left image, to calculate the corresponding epipolar line in the right image. In order to reduce the time required to find these correspondences, it is possible to transform the images in such a way that the epipolar lines are parallel and horizontal. Thus, each corresponding pixels in the transformed images are on the same pixel line. This transformation reduces the research space from 2D to 1D [87].
The image transformation process is called *rectification* and is reached with the application at run-time on both stereo images of *rectification matrix*, output of *calibration* phase.
Camera calibration defines the pose estimation of a camera in terms of interior and exterior orientation parameters. The term pose estimation in computer vision refers to the determination of the camera position and orientation using correspondence of 3D reference points.

This task can be accomplished using a technique computing the camera position and orientation with regards to a known object using 4 or more coplanar feature points. Commonly, in order to have artificial coplanar feature points, the known object is represented by a chessboard, but it is also possible to extract natural feature points from images with a scale-invariant feature transform algorithm.
The higher is the number of extracted features, the greater will be the accuracy of the dense map, but, at the same time, the higher will be the time to compute the calibration.

2.6.1.2 Dense map computing

This second real-time phase is composed of different steps: (i) application of rectification matrices, (ii) search and match of features on epipolar lines, and (iii) distance estimation.

- *Rectification*
  This first mandatory step aims at projecting the two stereo images onto a common image plane. It corrects image distortion by transforming the image into a standard coordinate system. This translation is performed exploiting the rectification matrices provided by the calibration step. Applying this matrix to the two stereo images, epipolar lines associated with matching pixels will be parallel and horizontal. In general this image transformation involves simple operations, but it is a time-consuming step, since it requires to move pixels taking them from one place in the original image and locating them in another position in the new rectified image.
  For the sake of completeness, Figure 2.27 shows an example of image rectification.

- *Search and match of features*
  This second step aims at finding the matches between the features extracted from the left and right images. Literature proposes a series of methods to solve this problem that can be grouped in: *feature-based* and *block-based*.
  The former methods extract characteristic points in the images (e.g., corners, edges, etc), called features, and then try to match the extracted features between the two acquired images [182, 199]. Instead, block-based methods consider a window centered in a pixel in one of the images and determine correspondence by searching the most similar window in the other image [31, 147].
  Nowadays, the research community focused more on block-based methods. They provide a complete, or semi-complete, dense map, while feature-based methods provide depth information of some points, only [38]. However, block-matching algorithms [38, 206, 217, 243] being, in general, more complex than feature-based algorithms, lead to longer execution times.

(a) Input stereo images

(b) Rectified stereo images

Figure 2.27: Image rectification example

- *Distance Estimation*

  This steps aims at locating the matched features in the 3D space. The Distance between the observer (i.e., two cameras composing the stereo-vision system) and the feature location in the 3D space can be computed exploiting the dense stereo approach.

  Starting from the corresponding features, or windows, and the stereo camera parameters, such as the distance between the two cameras and their focal length, the depth map can be extracted trough triangulation [14].

  Looking at Figure 2.28, knowing the focal length of the two cameras, the depth D_i of a feature point P_i, i.e., the distance between the point and the baseline b of the stereo camera, can be computed as:

$$D_i = \frac{f \cdot b}{x_{1,i} - x_{2,i}} \tag{2.14}$$

  where $x_{1,i}$ and $x_{2,i}$ represent the x-coordinates of the considered matched feature P_i in the two acquired images, and f is the *focal length* of the two cameras.

  From these information, the dense map can be computed, in which points closer to the camera are almost white whereas points further away are almost black. Points in between are shown in gray-scale, which get darker the further away the point gets from the camera. Figure 2.29 shows an example of dense map.

  As can be noted from te reported example, this method is able to provide accurate results

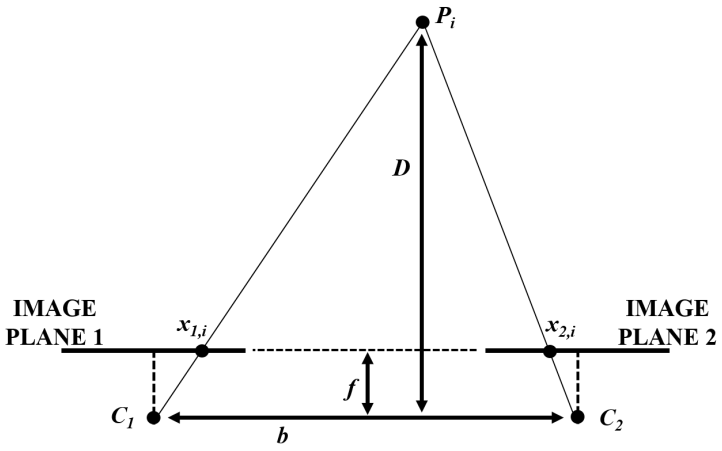
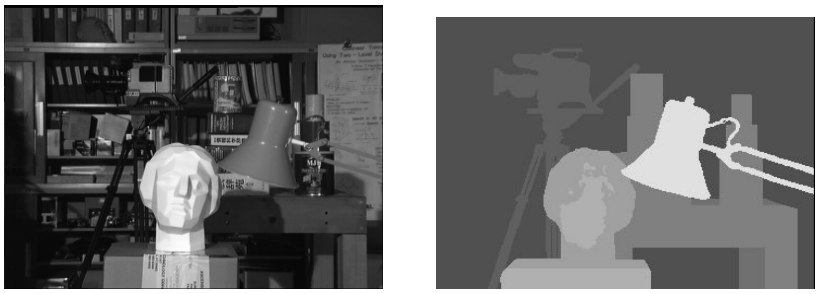


Figure 2.28: Trigonometric for Distance Estimation from Disparity



(a) Input image (b) Dense map

Figure 2.29: Dense map example

just in the points of the image characterize by a lot of texture, or, in general, strong contrast variation. Instead, the part of the object that are mainly textureless are considered at the same distance, resulting in a flat object surface.

2.6.2 Shape-from-Shading

When dealing with monochromatic and texture-less objects, under several assumptions, *Shape-from-Shading* (SfS) algorithms are the most recommended [92]. Contrary to stereo-vision methods, SfS algorithms exploit information stored in pixel intensities in a single image. Basically, these algorithms deal with the recovery of shape from a gradual variation of shading in the image, that is accomplished by inverting the light reflectance law associated with the surface of the object to be reconstructed [92].

Commonly, SfS algorithms assume as reflectance model the Lambertian Law [238]. In the last

years, algorithms based on other more complex reflectance laws (e.g. *Phong model* [239], *Oren-Nayar model* [166]) have been proposed. However, the introduced complexity requires a high computational power, leading to very low performances in terms of execution times. Nonetheless, the surface of the biggest debris (e.g., Ariane H10 stage) are characterized by an almost uniform surface, that can be effectively modeled with the Lambertian model. For these reasons, in the following we will focus on the analysis of the most important SfS algorithm based on the Lambertian law.
These SfS algorithms can be classified in three main categories: (i) methods of resolution of Partial Differential Equations (PDEs), (ii) optimization-based methods, and (iii) methods approximating the image irradiance equation [60, 238].

The first class contains all those methods that receive in input the partial differential equation describing the SfS model (e.g., *Eikonal Equation* [92]), and provide in output the solution of the differential equation (i.e., the elevation map of the input image).
In the years, several approaches of resolution, based on different mathematical tools, have been tested. The *Characteristic Strips Expansion* one [91] considers the shape to be reconstructed as an integral along a set of convergent straight lines. However, this proposed assumption is valid iff the surface of the object to be reconstructed follows fixed photometric characteristics. Moreover, these approaches face two other main issues. The former concerns the inherent defect of error accumulation, that is typical of every method of integration-based resolution approach. Instead, the latter concerns the boundary conditions on which the solution starts (i.e., the determination of the characteristic lines).
Many other approaches base the solution of the SfS differential equation model on the *Power Series Expansion* [62]. These methods start from the assumption that the brightness function associated with the object surface is analytical. Since real images are not analytical, this assumption makes this algorithms effective on synthetic images, only.
The *Level Set Method* [114] splits the solution of the SfS model into two steps. First, it performs an initialization phase in order to define the singular points (i.e., points with locally minimal depth), and then this partial solutions are merged according to certain rules based on differential geometry to obtain the global solution. Among these methods, the most effective ones are called *fast marching methods* [245]: they allow to accurately reconstruct the shape, but require a very high execution time.
Eventually, the approaches based on the *Approximation of Viscosity Solutions* (AVS) [61] are able to properly solve the SfS model, but they do not provide always the maximal solution (i.e., this leads to the well-known convex-concave ambiguity problem [29]). In order to overcome this issue, Sagona et al. in [61] ensures to find the maxima solution by minimizing an a priori defined error function. Basically, this approach iterates until the error function is lower than a fixed threshold.

The *Optimization-based* methods include all the algorithms that compute the shape by min-

imizing an energy function based on some constraints on the brightness and smoothness [238]. Basically, these algorithms iterate until the cost function reaches the absolute minimum [194]. However, in some of these algorithms, in order to ensure the convergence and reduce the execution time, the iterations are stopped when the energy function is lower than a fixed threshold [43], not guaranteeing accurate results.

Finally, the third class includes all the SfS methods that make an approximation of the image irradiance equation at each pixel composing the input image. These methods, thanks to their simplicity, allow to obtain acceptable results, requiring a limited execution time [3].

2.7 Image compression

In the last years the number of cameras mounted on satellites, spacecrafts and rovers increased more and more, leading to a higher number of images to be sent on ground. The first impact of this is the need of a higher bandwidth, storage space, and transmission time in order to deal with the increased amount of data. A more and more adopted solution for reducing the data size is the usage of data compression algorithm. Thus, the need for data compression, and, in particular, for image compression, has increased at a very high rate.

Image compression can be classified into two types: *Lossy* and *Lossless* compression techniques.
In *Lossless compression* there is no information loss and the image can be decompressed obtaining exactly the original one. Main applications are Medical imagery, video-based navigation, and drive assistant.
Instead, in *Lossy compression* loss and missed information is bearable, so, the output of these algorithms is more suitable for application that does not require to extract peculiar and precise information from the compressed image (i.e., features, edges, etc.).
Comparing the provided performance, from the one hand lossless methods cannot provide enough compression ratio (i.e., the ratio between the size of the original image and the compressed one), from the other hand, lossy methods can reduce to much the quality of the image making it useless.

Another possible classification takes into account the information on which the image compression is performed. In other words, if the image compression is performed considering the information contained in the current image, only (*Intra-Frame Compression*), or in a set of previous images, as well (*Inter-Frame Compression*).
The *Intra-Frame Compression* algorithms are exploited in all those fields in which a frame stream is not available (e.g., earth observation and digital picture archive). Example of these image compression algorithms are: JPEG-LS [101], JPEG2000 [198], FELICS [93], CALIC [223], and CCSDS-LDC [204].
Instead, *Inter-Frame Compression* category involves all the algorithms used in video compres-

sion, like H.264 [175], MPEG4, and AVCHD [189]. Comparing the maximum compression rate of these two algorithm families, the *Inter-Frame Compression* family provides a compression rate extremely higher than the *Intra-Frame Compression* one. However, this advantage strongly decrease with a decreasing frame rate (i.e., the number of frame composing a second of video stream). In particular, the threshold to decide which kind of algorithm family prefer is commonly set to 10 frame-per-second (fps).

Since the pictures commonly acquired by satellite, spacecraft, and rovers that are useful for scientific analysis are single-shot image or occasionally image stream with a low frame rate, the most important and exploited family of image compressor is the *Lossless Intra-Frame Compression*.

In the last years a lot of studies have been performed to compare the performance of the *Lossless Intra-Frame Compression* algorithms [184] [219] [234]. Analyzing in details these studies, it is possible to depict that the algorithm that provide the highest compression rate, for both color or gray-scale images, are the JPEG-LS, JPEG2000, and CCSDS-LDC.

Another important parameter to select the best algorithm to compress images acquired by satellite, spacecrafts, and rovers is the complexity. This parameter allows to quantify the required resources to implement an algorithm, that are limited in all space missions and in particular in space exploration missions (e.g., Mars mission). The complexity has been evaluated comparing the area occupation of the current state-of-the-art implementations associated with the aforementioned best Intra-frame lossless algorithms. Table 2.2 reports the area occupation in terms of Logic Elements (LEs, i.e., a logic element in an FPGA/CPLD device is defined as a storage element (Flip-Flop), plus a Look-Up Table (LUT)), and Digital Signal Processors (DSPs). For the sake of completeness, in Xilinx Virtex 4 FPGAs (Section 3.3) each Slice is composed of two Logic Elements.

Table 2.2: Lossless Intra-Frame area occupation comparison

Algorithm	**LEs**	**DSPs**
JPEG2000 [10]	13,500	-
CCSDS-LDC [235]	11,806	8
JPEG-LS [113]	8,616	-

In Table 2.2 the area occupation of the algorithms that can be efficiently implemented on an FPGA device. As one see, the lowest area occupation (i.e., complexity) is provided by the *JPEG-LS*, that demonstrates that its low algorithm complexity can be directly translated in low hardware resources usage. This low resources usage is of particular importance in space exploration missions, in which the constraints in terms of resources usage and weight are really stringent.

Another important feature of this algorithm is the near-lossless compression. This last pe-

culiarity ensures to act on the input parameters in order to increase more and more the compression rate while introducing a loss of information that is lower than the one introduced by the lossy compression algorithms (Section 5.3.2).

The rest of this chapter deeply describes and analyzes the operations required by the JPEG-LS algorithm (Section 2.7.1).

2.7.1 JPEG-LS Algorithm

JPEG-LS is the latest ISO/ITU-T87 standard for lossless/near-lossless coding of still images. The main procedures for the lossless and near-lossless encoding process are shown in Figure 2.30.

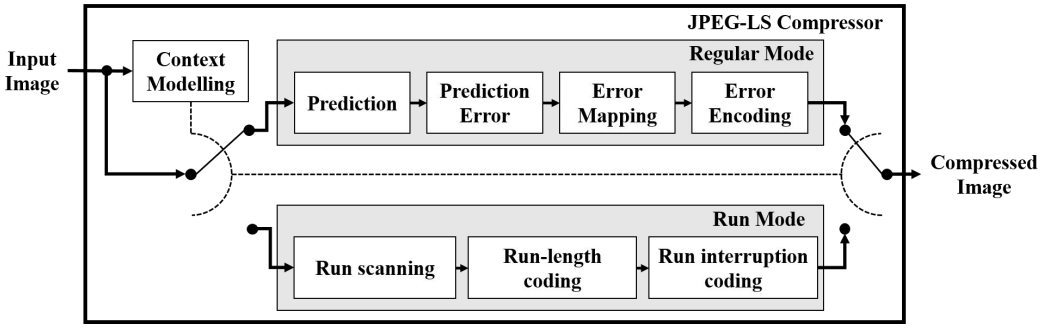


Figure 2.30: JPEG-LS main procedures

This compression algorithm is composed of three main procedures: (i) *Context Modeller*, (ii) *Regular Mode*, and *Run Mode*.

The *Context Modeller* defines if the pixel to be compressed belongs to a flat image region or not. A flat image region is define as a region of equal pixels, for the lossless compression, or a region of pixels with a maximum difference in the intensity value equal to $NEAR$, in the near-lossless compression. The $NEAR$ parameter allows to tune the trade-off between the achieved compression level and the introduced loss of information in the input image. In fact, from the one hand, since the compression of a flat region ensures to reach higher compression level [101], increasing the value of the $NEAR$ parameter bigger flat image regions can be found, and so the size of the compressed image can be strongly reduce. On the other hand, since all pixels belonging to a flat region are decompressed with the same intensity value, a higher $NEAR$ parameter value leads to embed in the same flat image region pixels with highly different values, and so increase the error on the decompressed image.

The *Context Modeller* to efficiently define if a pixel start a flat region makes use of adjacent pixels to the current one. In particular, as shown in Figure 2.31, the exploited adjacent pixels are the ones in the west (Ra), north-west (Rc), north (Rb), and north-east (Rd) position with respect to the current pixel to be encoded (Ix).

In particular, these adjacent values are used to compute local gradients $D1$, $D2$, and $D3$ as in

	Rc	Rb	Rd	
	Ra	Ix		

Figure 2.31: Reconstructed samples position

Equation 2.15, 2.16, and 2.17, respectively.

$$D1 = Rd - Rb \tag{2.15}$$

$$D2 = Rb - Rc \tag{2.16}$$

$$D3 = Rc - Ra \tag{2.17}$$

If the local gradients are all zero, for lossless compression, or their absolute values are less than or equal to the $NEAR$ parameter, for the near lossless compression, Ix is consider the starting pixel of a flat region and *Run mode* procedure is activated, otherwise the *Regular mode* one is activated.
Moreover, before starting the execution of the selected procedure, the *Context Modeller* quantizes $D1$, $D2$, and $D3$. The quantization provides in output a quantized context for each local gradient (i.e., $Q1$ for $D1$, $Q2$ for $D2$, and $Q3$ for $D3$).
These three values are mapped, on a one-to-one basis, to produce the final context value ($context$) associated with Ix. The mapping procedure is not defined by the standard, but the final value of the context must be in the range [0:364].
In the C/C++ implementation of the JPEG-LS, that can be freely downloaded from [102], it is proposed to speed-up the context quantization and mapping by mean of four Look-up Tables (LUTs): $vLUT1$, $vLUT2$, $vLUT3$, and $ClassMapvLUT$. In particular, the $vLUTx$, is pointed by the Dx value to obtain the Qx context, where x can be 1, 2, or 3.
The sum of the three quantized contexts (Q_sum) is used to point the $ClassMapvLUT$ to obtain $ClassMapcontext$, that is used to compute the final value of $context$.

As aforementioned, the *Regular Mode* module operates just when Ix is not the starting pixel of a flat image region. This module internally stores four statistic parameters for each possible context value (i.e., each one of these parameter is a 365 cells vector, since the context value can be in the range [0:364]). These parameters are: the accumulator of the magnitudes of previous prediction errors ÎŚ, the bias variable B, which accumulates the error values, the prediction error

correction parameter C, and the context occurrences counter N.
The *Regular Mode* module, to define the compressed bitstream associated with Ix, performs four consecutive operations. First, it executes the *Prediction* task, in which a median edge detector is used to predict the value of the current pixel (Px). As show in Equation 2.18 this operation is done exploiting the adjacent pixels used for the gradients computation (i.e., Ra, Rb, Rc, and Rd).

$$Px = \begin{cases} min(Ra, Rb), & \text{if } Rc \geq max(Ra, Rb) \\ max(Ra, Rb), & \text{if } Rc \leq min(Ra, Rb) \\ Ra + Rb - Rc, & \text{otherwise} \end{cases} \tag{2.18}$$

This operation is followed by the *Prediction Error* task, in which Px is corrected using the value of the bias cancellation variable C pointed by $context$. Moreover, the error value is clamped in an appropriate range to increase the efficiency of the coding process.
Before the actual compression task, the prediction error is mapped (i.e., *Error Mapping* task in Figure 2.30) by performing modulo reduction and quantization operations.
The *Regular Mode* module ends its operations by encoding the mapped prediction error (i.e., *Error Encoding* task in Figure 2.30) to define the compressed bitstream associated with Ix. This encoding procedure is done using the *Golomb coding* [80]. This kind of coding technique has been selected since it is really efficient on small values, like the ones associated with the mapped prediction error.
First, the Golomb coding variable k is computed according to the A and N statistical parameters, as shown in Equation 2.19

$$for(k = 0; (N[context] << k) < A[context]; k++) \tag{2.19}$$

where $context$ is the final context value defined by the *Context Modeller*.
Then, the mapped prediction error is encoded using an optimal codes for a *Two Sided Geometric Distribution* (TSGD) [138], based on *Golomb codes*, and the defined compressed bitstream is provided in output. Eventually, the *Regular Mode* module updates the values of the A, B, C, and N statistical parameters associated with $context$.

The *Run Mode* procedure operates when Ix is the starting pixel of a flat image region. This procedure computes three consecutive operations. First, it defines the length of the current flat region executing the *Run scanning* task in Figure 2.30. This task requires to run inside the Ix image row until the end of the flat image region is found. Basically, it reads subsequent pixel values, in the same image row, until it finds a pixel value different from Ix, in the lossless compressing, or a pixel whose difference with Ix is higher than $NEAR$, in the near lossless compression. This pixel value is called *Run interruption sample*, and it identifies the end of the current image flat region. It must be precise that if this pixel value has not been found when the end of the current image row is reached, the flat image region is anyway considered concluded.

The *Run Mode* module defines the compressed bitstream to be provided in output by encoding both the defined flat image region size and the *Run interruption sample*. The former is encoded exploiting an efficient technique, called *block-MELCODE encoding* [165]. Instead, the latter is compressed using the *Golomb coding* as in the *Regular Mode* module. Eventually, also in this case, the the values of the A, B, C, and N statistical parameters are updated.

For more detailed information about the operations performed by JPEG-LS algorithm, the reader may refer to Appendix A.

CHAPTER 3

FIELD PROGRAMMABLE GATE ARRAY FOR SPACE: A FLEXIBLE WAY TO ACHIEVE HIGH-PERFORMANCE

In the last decades the increasing demand of computational intensive tasks performed at real-time performance makes the software execution on microprocessor inefficient. The only way to reach the required performance is the off-loading of these time-critical tasks on dedicated hardware devices, exploiting the so-called *hardware acceleration*.

The implementation of these hardware accelerators can be commonly done exploiting Application Specific Integrated Circuits (ASICs) [187] or Field Programmable Gate Arrays (FPGAs) [209].
ASICs are non-standard integrated circuits that are designed for a specific use or application. These devices are designed starting from an architecture description in a Hardware Description Language (HDL) [79] that is directly synthesized, exploiting cell libraries and design rules, on silicon [1].
Instead, FPGAs are programmable devices that enable to skip the silicon design since the described architecture is synthesized to program an already verified and characterized device. Basically, an FPGA device internally embeds user logics (e.g., registers, look-up table, on-chip memory, arithmetic resources, etc.), plus a configuration layer that can be programmed by the designer to properly interconnect the internal logics and implement the desired architecture.

Due to these main differences, FPGAs and ASICs provide various values to designers, and they must be carefully evaluated before choosing any one over the other. While FPGAs used to be selected for lower speed/complexity/volume designs in the past, today's FPGAs easily push the 500MHz performance barrier. Moreover, they provide unprecedented logic density increases and a host of other features, such as embedded processors, DSP blocks, clocking, and high-speed serial at ever lower price points. For these reasons, FPGAs are a compelling proposition for almost any type of design.

FPGAs and ASICs offer different benefits to the designer. From the one hand, ASICs provide

full custom capability, since these devices are designed from the scratch starting from system specifications, and, consequently, free choices on the desired form factor (i.e., the size, configuration, or physical arrangement of a hardware object). From the other hand, FPGAs ensure a high flexibility, fast time-to-market, and low Non-Recurrent-Engineering (NRE) costs.

The NRE costs are extremely lower for the FPGA technology for two main reasons. First, FPGAs require a simpler design cycle, removing the complex and time-consuming floorplanning, place and route, timing analysis, and mask/re-spin stages of the project since the design logic is synthesized to be placed onto an already verified and characterized device. Second, they completely avoid foundry costs associated with the mask development, since the silicon of the FPGA device is already designed and it has just to be programmed with the desired architecture.
Due to this high NRE cost, ASICs can reach competitive single device price just for high-volume production. Instead, FPGAs are preferred when just few devices must be produced.
Since in space applications a custom hardware accelerator is commonly designed for just one or few missions, in the last decade space agencies start to prefer FPGAs instead ASIC for different applications and missions.

Moreover, as stated in [77], thanks to the large amount of hardware resources embedded in modern FPGAs, digital designers can use these components to perform not only familiar logic functions, but also tasks that were formerly handled at the board level by separate, dedicated parts. Large re-programmable devices eliminate the need for components such as Phase Lock Loops (PLLs), voltage translation buffers, and memories when on-chip storage resources are not sufficient. This high level of integration allows designers to reduce overall system power requirements and cut costs.

However, since FPGA devices are highly susceptible to *Single Event Upset* (SEU) [215], not all FPGA technology can be directly used in space environment. Actually, three space-qualified FPGA technologies are available:

- *One-Time-Programmable FPGAs*
- *Flash-based FPGAs*
- *SRAM-based FPGAs.*

These three technologies are deeply described and analyzed in Section 3.1, 3.2, and 3.3, respectively. Moreover, Section 3.4 briefly describe techniques to hardened design implemented in FPGA against SEUs.

3.1 One-Time-Programmable FPGAs

One-Time-Programmable (OTP) FPGAs are devices that can be just programmed once and then are able to maintain the configuration also at power-off. This is possible since in these devices

the configuration is hold by an antifuse metal layer.

The unique family of OTP FPGAs that is space-qualified is the *Microsemi RTAX-S/SL* [41] [141]. In these FPGAs the user logic and the configuration layer are split in two different layers. In particular, the configuration layer is a metal-to-metal antifuse programmable interconnect element that resides between the upper two layers of metal (Figure 3.1). The antifuse connections are

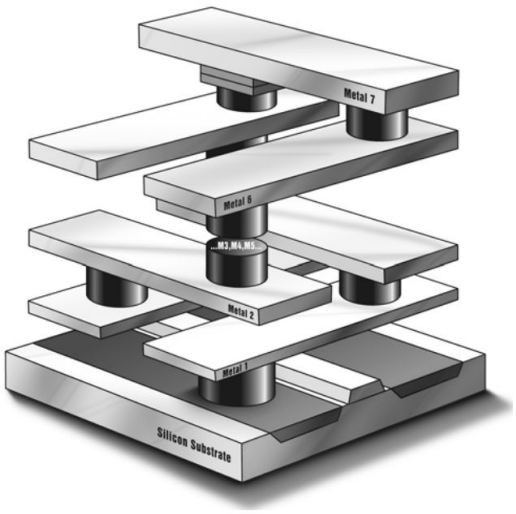


Figure 3.1: Configuration layer in *Microsemi RTAX-S/SL* FPGAs [141]

usually open and when are programmed compose a low-impedance interconnection among the different metal layers. The low impedance nature of this interconnection ensures to reach very high communication speed.
Moreover, the metal-to-metal antifuse programmable interconnect element does not occupy silicon area in the user logic layer, making possible the *sea-of-modules* architecture. As shown in Figure 3.2, this kind of architecture ensures to fully exploit the silicon area without wasting space for the routing resources.

In particular, the user logic layer is composed of two basic elements: *R-cell* and *C-cell*.

The former is a register hardened against SEUs triple redundancy to achieve a *Linear Energy Transfer* (LET) threshold of greater than $60 MeV - mg/cm^2$ [141]. The triplication is obtained by inserting three master-slave latch pairs in which the three outputs of the master and slave side are majority voted in order to apply the fault tolerant technique. In order to avoid fault accumulation, each master-slave latch is equipped with a feedback that, in case of fault, restore the correct value in the latch. Moreover, during the floorplan of the user logic layer, the location of the three pairs of master-slave latches is defined to avoid that a single ion induces a SEU in more than one latch. For the sake of completeness, Figure 3.3 shows the difference between a standard R-cell (Figure 3.3(a)) and its SEU-hardened version (Figure 3.3(b)).

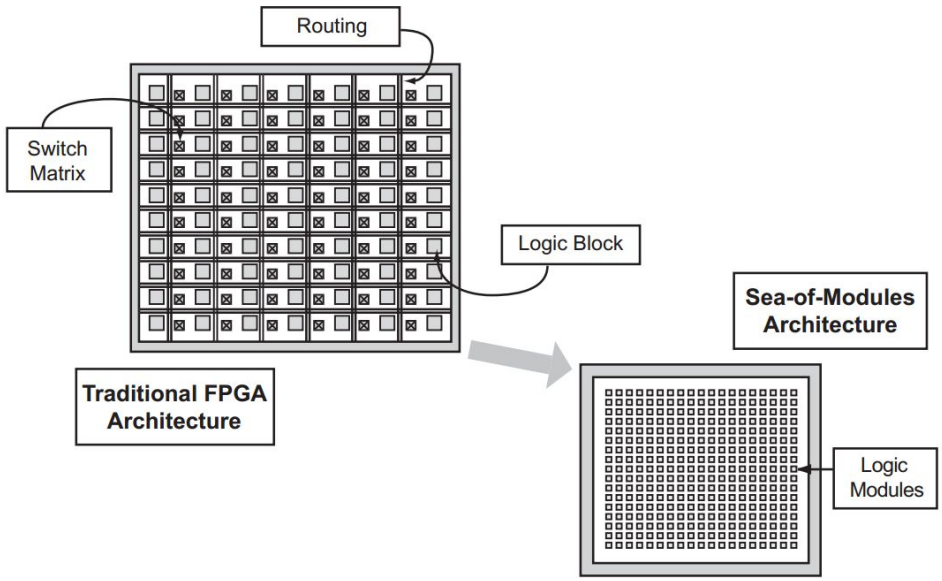
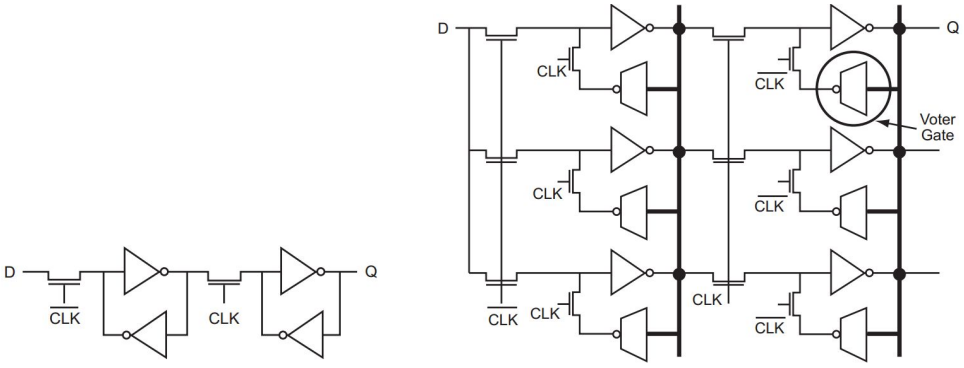


Figure 3.2: Comparison of the user logic layer between common FPGA and sea-of-modules architectures [141]



(a) Standard R-cell (b) SEU-hardened R-cell

Figure 3.3: R-cell comparison [141]

The C-cell is a combinatorial block able to implement up to 4,000 5-input combinatorial functions. In addition, this cell is equipped with a dedicated carry logic for improving the performance of arithmetic functions.

Internally to the *Microsemi RTAX-S/SL* FPGA, an R-cell is packed together with two C-cells, two Transmit (TX), and two Receive (Rx) routing buffers inside a *Cluster*. A couple of Clusters composes a *SuperCluster* (Figure 3.4). Moreover, each SuperCluster contains a *Buffer* used to minimize system delays when high fanout logic functions are implemented combining different

SuperClusters.

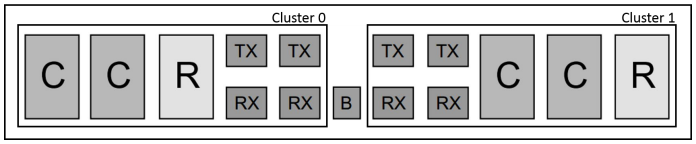


Figure 3.4: SuperCluster internal architecture [141]

Thanks to the modularity of the internal architecture, during synthesis the user defined architecture can be fine-grained split among SuperClusters and their internal elements, allowing to reach high performances and a fast design flow.

Figure 3.5 shows the chip-level architecture of *Microsemi RTAX-S/SL* FPGAs. It can be noted

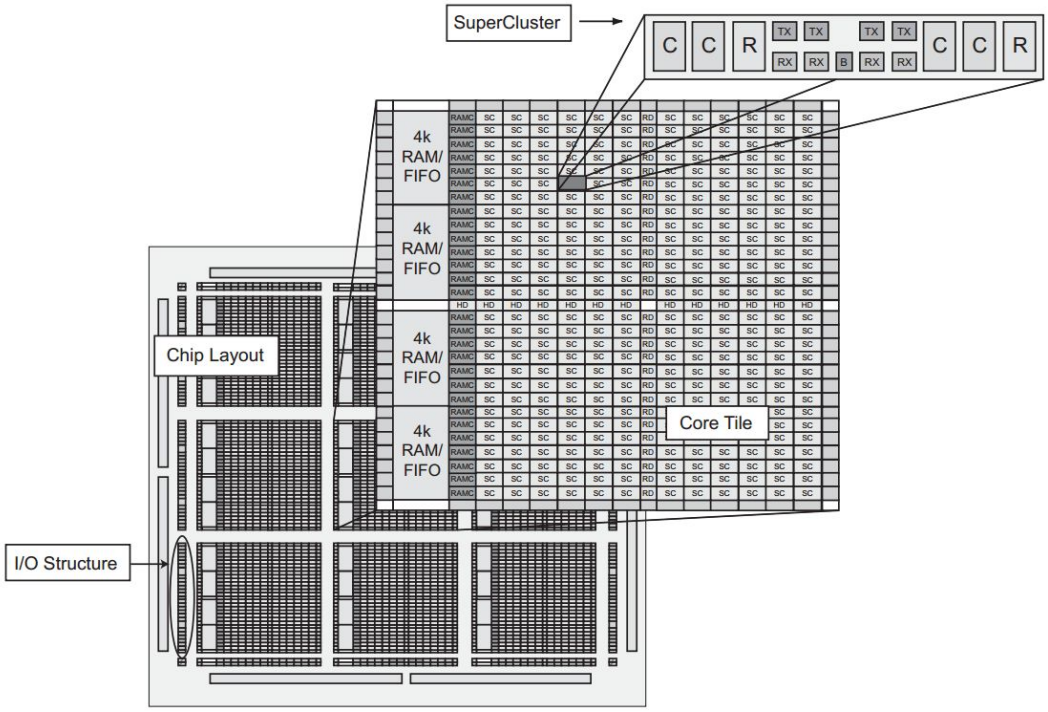


Figure 3.5: Example of an *Microsemi RTAX-S/SL* FPGA chip level architecture [141]

that the chip is composed of different *Core Tiles*, each Core Tile contains a lot of SuperClusters, plus a certain amount of SRAM blocks. The number of SuperClusters and SRAM blocks change depending on the selected FPGA, more detailed information are provided in [141].

Each SRAM block is 4,608 bits in size. However, the designer can freely configure these storage elements, both in terms of aspect ratio and read/write port width, depending on its needs. Moreover, each SRAM block contains a *First In First Out* (FIFO) control unit [201], allowing the design

of FIFO buffers without wasting C-cells inside SuperClusters. As well as SRAM blocks, a FIFO can configured in terms of depth (if the required depth exceeds the storage elements contained in a single SRAM block, the FIFO is implemented concatenating different blocks) and control signals (i.e., *EMPTY*, *ALMOST EMPTY*, *FULL* and *ALMOST FULL* [201]).

Since SRAM blocks are not rad-hardened by design, if the developed system must work in an environment characterized by high energy particles, two mitigation techniques can be easily applied. The former is an Error Detection and Correction IP-core (*CoreEDAC* [140]), already developed by Microsemi, that can be directly connected to the storage element to be protected. The latter is a scrubbing IP-core, integrable in the aforementioned CoreEDAC IP-core, that ensures the data integrity while the memory block is not used through a memory scrubbing approach [183].

Eventually, the chip-level architecture of *Microsemi RTAX-S/SL* FPGAs (Figure 3.5) is completed by *I/O structures*. The flexible nature of I/O structures allows designers to apply mixed-voltage configurations of input and output pins. Each I/O structure can be programmed to work in different operational modes, i.e., single-ended, differential-ended and voltage-referenced. This flexibility enables to support a lot of different communication standards.
The I/O structures are organized in *I/O clusters* (Figure 3.6), each containing two *I/O modules*, four RX modules, two TX modules, and a buffer (B) module [141]. Each I/O module is composed of an input register, an output register, and an enable register. All of these registers are hardened against SEU with the same mitigation technique used in R-cell [141].

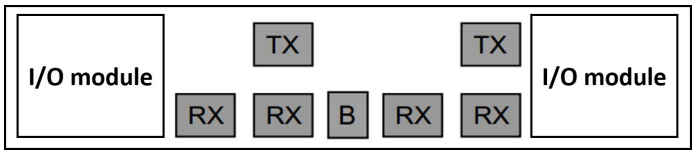


Figure 3.6: I/O cluster internal architecture

After the internal architecture analysis, it can be noted that *Microsemi RTAX-S/SL* FPGAs provide a lot of already implemented fault-mitigations techniques against SEUs. For these reasons in the last years this kind of FPGAs have been extensively used by space agencies both on satellite, working in the Low Earth Orbits (LEOs), and spacecraft, involved in space exploration missions. Figure 3.7 shows some examples of the missions in which these FPGAs have been used.

3.2 Flash-based FPGAs

Flash-based FPGAs, differently from OTP FPGAs (Section 3.1), are reprogrammable devices in which the configuration is held by a flash-based infrastructure. Basically, the user logic contained in the FPGA is interconnected to obtain the desired architecture through a routing network implemented with flash-based switches [236]. Since the configuration is stored exploiting a flash

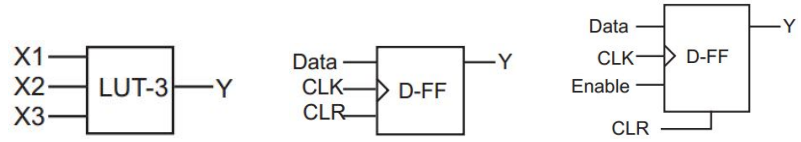
Figure 3.7: Missions explointing *Microsemi RTAX-S/SL* FPGAs

technology, this kind of FPGA is able to be *Live At Power-Up* (LAPU). In other words, the FPGA can be switch-off without losing the internal configuration.

The unique vendor that offers space-grade flash-based FPGAs is Microsemi with the *Microsemi RT ProAsic* FPGA family [40] [139]. In this FPGA family the configuration is stored resorting to an advanced flash-based 130 nm LVCMOS process [202] with 7 layer of metal.
A key aspect of the *Microsemi RT ProAsic* FPGAs configuration infrastructure is the so-called *Flash*Freeze* technology [139]. This technology is able to completely shut down the dynamic power consumption, while retaining FPGA memories and registers content, without the need of switching off the system clock and related distribution network. To ease the power management in the design, the user can quickly switch on and off the *Flash*Freeze* mode (i.e., in less than 1 μs) by driving a dedicated FPGA pin. Thanks to this features, this kind of FPGAs is able to reduce the dynamic power consumption by 40% and static power consumption by 50% w.r.t. other FPGA technologies.

The user logic in *Microsemi RT ProAsic* FPGAs, differently from the configuration infrastructure, is implemented using standard CMOS technology. The basic element composing the user logic is the *VersaTile* [139]. Each VersaTile, as shown in Figure 3.8, can be configured in order to implement: all 3-input combinatorial functions, a latch with clear or set, a D-type flip-flop with clear or set, or Enable D-type flip-flop with clear or set.
Moreover, each VersaTile is equipped with nine *VersaNets*. A VersaNet is an element allowing a minimum skew distribution of the clock signal, or a minimum delay communication among VersaTiles composing a combinatorial network with high fanout.

Figure 3.9 shows the chip-level architecture of *Microsemi RT ProAsic* FPGAs. It can be noted



(a) 3-input combinatorial function

(b) D-type flip-flop

(c) Enable D-type flip-flop

Figure 3.8: VersaTile configurations [139]

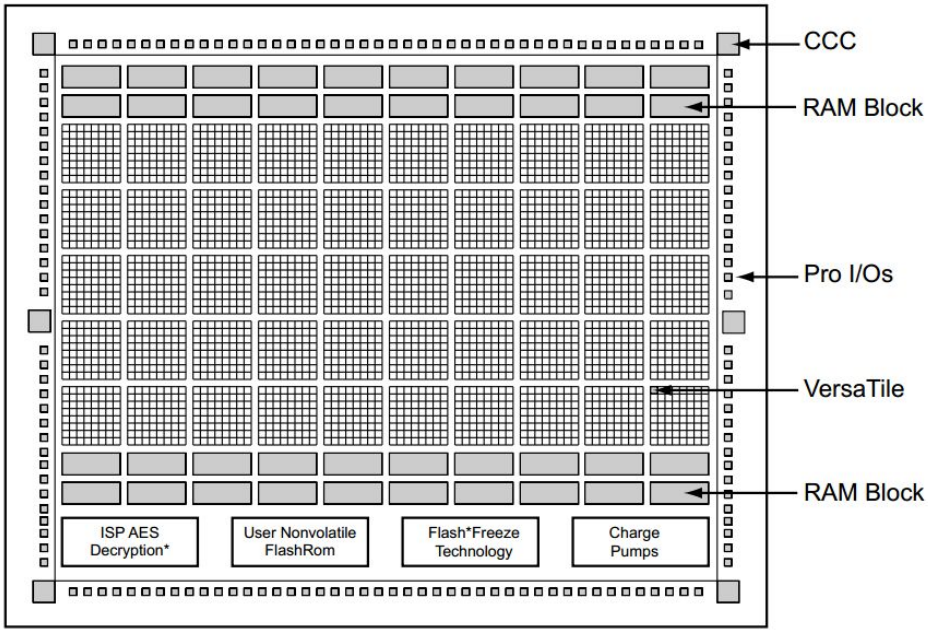


Figure 3.9: Chip-level architecture of *Microsemi RT ProAsic* FPGAs [139]

that in addition to VersaTiles and the management unit for the *Flash*Freeze* technology, *Microsemi RT ProAsic* FPGAs contains:

- Clock Conditioning Circuitries (CCCs)
- Advanced I/O modules (Pro I/Os)
- RAM blocks
- User non-volatile Flash ROM.

Six CCCs are contained in every *Microsemi RT ProAsic* FPGA. Each CCC is equipped with a configurable Phase Locked Loop (PLL) [11], that allows designers to generate a custom clock frequency starting from an input clock reference. The frequency of the generated clock signal must

be in the 0.75-250 MHz range, while the input input clock reference in the 1.5-250 MHz range. From these data it is possible to understand that the maximum operating frequency of a design implemented in the *Microsemi RT ProAsic* FPGA is 250 MHz. Moreover, CCCs are equipped with sophisticated control circuitry ensuring to minimize the jitter and vary the phase of the generated signal. Eventually, the location of CCCs (four located at the FPGA corners, and two at the mid left and right FPGA side, see Figure 3.9) minimizes the skew associated with the clock distribution.

Advanced I/O modules provide a flexible management of I/O interfaces. In fact, they supports different voltages (i.e., 1.2, 1.5, 1.8 and 2.5 V), and a plenty of I/O standards (i.e., single-ended, differential, voltage-referenced), enabling a great adaptability of these FPGA to different communication links.

RAM blocks provide on-chip memory resources to the designer. These blocks can be programmed to support a variable aspect-ratio, that must not exceed the available storing resources in a bank (i.e., 4,608 bits). Each module has an independent read and write port in which the port width can be independently configured. Moreover, RAM blocks are equipped with a FIFO control unit that enables the implementation of a FIFO buffer without wasting VersaTiles. This control unit is fully customizable to implement different FIFO lengths and FIFO control signals configurations. Obviously, to implement high sized FIFO buffers different RAM blocks can be cascaded.

User non-volatile Flash ROM is a peculiar resource of *Microsemi RT ProAsic* FPGA (i.e., FPGAs based on other technology do not contain this kind of resource). This kind of resource can be really useful for designers especially when modules composing the developed system requires fixed configurations (e.g., Internet protocol addressing, Device serialization and/or inventory control, etc.). The Flash ROM size is limited to 1 kbit, and its content can be just modified at design time through a *Joint Test Action Group* (JTAG) programming interface, but can be read back via direct FPGA core addressing.

By comparing the internal architecture of *Microsemi RT ProAsic* and *Microsemi RTAX-S/SL* FPGAs, it can be immediately noted that the former do not provide SEU-hardened components. For these reasons, designers are in charge of applying SEU mitigation techniques (Section 3.4) depending on the flight-critical nature of the application.

Table 3.1 compares the available resources between the *RT3PE3000L* [139] and *RTAX4000S/SL* [141]. These two FPGAs are the biggest devices of the *Microsemi RT ProAsic* and *Microsemi RTAX-S/SL* FPGA families, respectively. In order to perform the comparison of the available user logic, the total amount of R-cells and C-cells in the *RTAX4000S/SL* must be sum, since each VersaTile of the *RT3PE3000L* can be configured both as a register or combinatorial function. Moreover, it must be noted that C-cells are more efficient than VersaTile when configured as combinatorial function, since the former are able to implement 5-input combinatorial functions, while the latter just 3-input ones. After these considerations, it can be depicted that the available user logic is almost the same in the considered devices. However, some advantages in favour of the *Microsemi*

Table 3.1: Comparison of *RT3PE3000L* and *RTAX4000S/SL* internal resources

	RT3PE3000L	***RTAX4000S/SL***
User logic	75,264 (VersaTile)	20,160 (R-cells) 40,320 (C-cells)
Embedded RAM/FIFO	504 kbits	540 kbits
Embedded Flash ROM	1 kbits	-
Embedded Clock Control Circuitries	6	-
Maximum User I/Os	620	840

RTAX-S/SL family still remain, since the registers in these FPGAs are already SEU-hardened by design, while in *Microsemi RT ProAsic* FPGAs hardening techniques must be applied wasting user logic.
The Embedded RAM/FIFO resources are almost the same in both devices, and Microsemi, as for the space-grade OTP FPGAs (Section 3.1), provides an EDAC IP-core [140] to protect RAM blocks against particles induced effects. Thus, no differences on the volatile memory resources can be appreciated between the two considered FPGAs. The same consideration can be done for the I/Os.

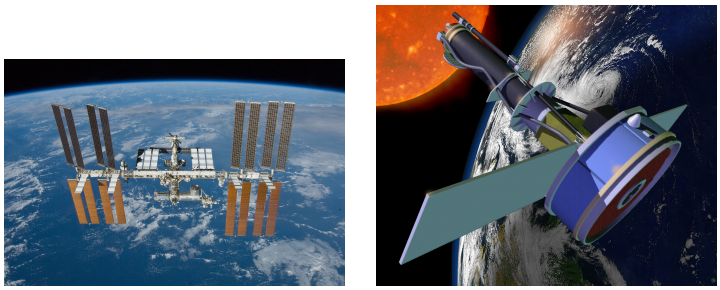
The main advantages provided by the *Microsemi RT ProAsic* FPGAs is the higher level of flexibility. In fact, differently from the *Microsemi RTAX-S/SL* family, *Microsemi RT ProAsic* FPGAs are reconfigurable and they embed configurable CCCs allowing multi-clock domains design without requiring external devices (e.g., PLLs). Moreover, as aforementioned, the *Microsemi RT ProAsic* FPGAs are the only FPGAs equipped with a Flash ROM module.

About the configuration layer, the flash-based technology exploited in *Microsemi RT ProAsic* FPGAs is immune to SEU effects, so no mitigation techniques must be applied on the configuration memory .

Thanks to the advantages concerning the flexibility, *Microsemi RT ProAsic* FPGAs are currently used on the *International Space Station* (Figure 3.10(a)) [153], and it has been successfully used in two NASA missions: the *Interface Region Imaging Spectrograph* (IRIS, Figure 3.10(b)) [152], and *LADEE Lunar Atmosphere and Dust Environment Explorer* (Figure 3.10(c)) [156]

3.3 SRAM-based FPGAs

Differently from OTP (Section 3.1) and Flash-based (Section 3.2), SRAM-based FPGAs are reprogrammable devices that lose their configuration when are powered-off, since the configuration layer is implemented through a volatile memory (SRAM memory). For this reason, to properly work these devices require an external non-volatile memory storing the configuration needed to program the FPGA at power-up.



(a) *International Space Station* (b) *IRIS*

(c) *LADEE Lunar Orbiter*

Figure 3.10: Missions exploiting *Microsemi RT ProAsic* FPGAs

The main vendor offering space-grade SRAM-based FPGAs is *Xilinx* [99]. Figure 3.11 shows the Xilinx space-grade FPGAs roadmap.

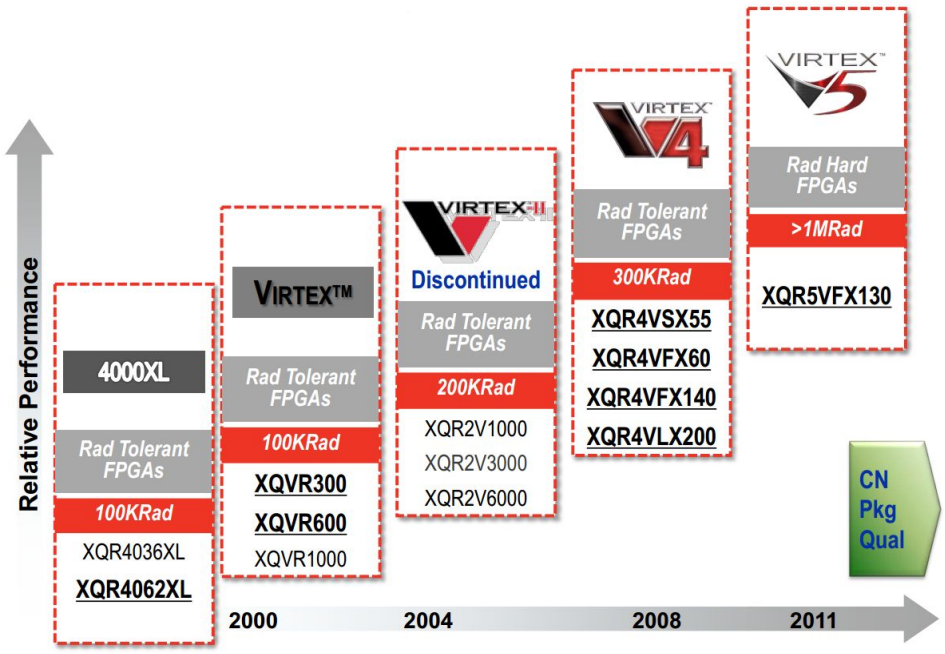


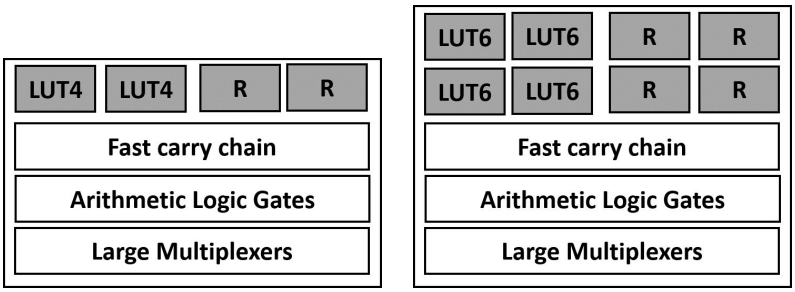
Figure 3.11: Xilinx space-grade FPGAs roadmap

As can be noted the actual state-of-the-art architectures are the *Xilinx Virtex 4 QV* [225], and *Xilinx Virtex 5 QV* [228] families.

The main difference between these two families is the Rad-Hardened By Design (RHBD) feature of the *Xilinx Virtex 5 QV* family. In *Xilinx Virtex 5 QV* FPGAs both the configuration memory and the internal registers (i.e., user logic and I/O registers) are rad-hardened by design against SEUs. Basically, the rad-hardness against SEUs is obtained exploiting dual-node latches approach that provide nearly 1,000 times the SEU hardness of the standard cell latches.

Another main difference concern the CMOS technology. In fact, *Xilinx Virtex 4 QV* FPGAs make use of the *90 nm Copper CMOS* process, *Xilinx Virtex 5 QV* FPGAs the *65 nm Copper CMOS* one. This difference allows to slightly boost up the speed performance in *Xilinx Virtex 5 QV* family from 400 MHz (i.e., the maximum operating frequency of *Xilinx Virtex 4 QV* FPGAs) to 450 MHz.

The basic elements composing the user logic in these two families is the *Slice*. Figure 3.12 highlights the differences in the Slice internal structure between the two families.



(a) *Xilinx Virtex 4 QV* Slice (b) *Xilnx Virtex 5 QV* Slice

Figure 3.12: Comparison of Slice internal structure

It can be noted that *Xilinx Virtex 4 QV* FPGAs embeds smaller Slices than *Xilinx Virtex 5 QV* FPGAs. Just two registers (Rs) and two Look-Up Tables (LUTs) are present in the *Xilinx Virtex 4 QV* Slice. Moreover, *Xilinx Virtex 5 QV* Slice provides 6-input LUTs (LUT6, i.e., these LUTs can be also configured as two 5-input LUTs, during synthesis, to increase the efficiency), while *Xilinx Virtex 4 QV* Slice just 4-input LUTs (LUT4). Concerning the other Slice resources the two family are equivalent.

Internally, slices are grouped in *Configurable Logic Blocks* (CLB). In *Xilinx Virtex 4 QV* family four Slices compose a CLB, instead in *Xilinx Virtex 5 QV* one just 2 Slice are required. Every CLB integrates internal fast interconnect and connects to access general routing resources.

The routing resources in space-grade Virtex families is provided through *General Routine Matrix* (GRM) or direct connection among CLBs. A GRM is an array of routing SRAM-based switches implemented in the aforementioned rad-hardened technology. Figure 3.13 shows the interconnections between GRM and CLB.

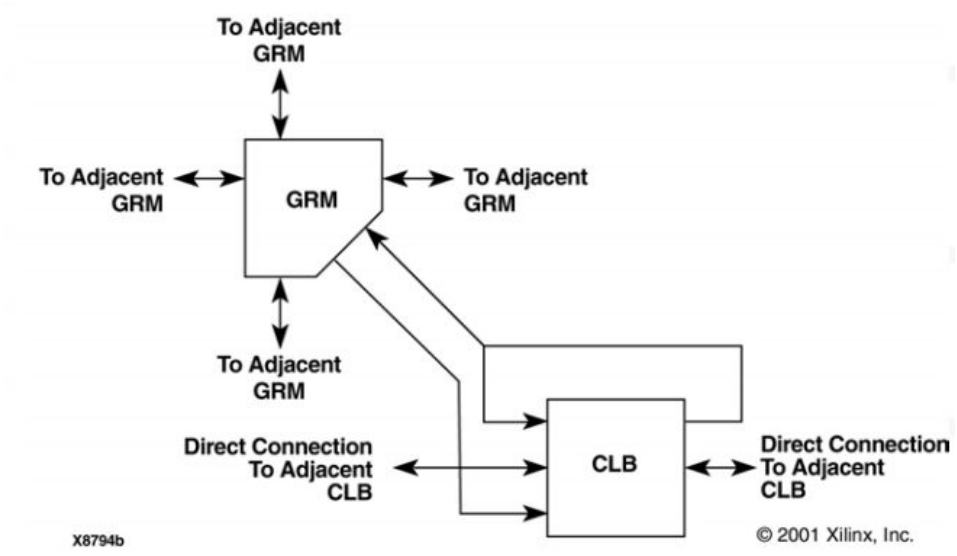


Figure 3.13: *Xilinx Virtex* FPGAs routing infrastructure

Each GRM provides connections to other GRMs through a North, East, South or West -side link, plus a direct connection to the closest CLB. Moreover, to increase the interconnection capability, CLBs are equipped with a direct connection to the left and right -side CLBs, plus a feedback connection to provide a loop-chain among internal LUTs. This routing infrastructure ensures a minimum delay for both short (through direct connections) and long (through GRMs) connections.

In addition to CLBs and GRMs, the internal architecture of space-grade *Xilinx Virtex* FPGAs is composed by four other components:

- *Block RAM* (BRAM)
- *Clock Management Tile* (CMT)
- *Extreme Digital Signal Processor* (DSP48E)
- *High-performance parallel SelectIO bank*.

BRAMs are configurable true dual-port SRAM memory blocks, so each memory port can be used in both write or read mode. Each BRAM is able to work up to 360 MHz, and is 36 kbit in size, but can be also used as two separate block of 18 kbit. The write/read port size can span from 1 bit width up to 36 bits width, for true dual-port memory, or 72 bits, for simple dual-port memory (i.e., one port used in write mode and one in read mode). BRAMs embed control logic to operate as FIFO buffer, also. This control unit is fully customizable to implement different FIFO lengths and FIFO control signals configurations. Moreover, each BRAM has a dedicated cascade

routing, that allows to reach high performance when more than one BRAM must be used to build a big memory block. Eventually, as in space-grade OTG and Flash-based FPGAs, there is an Error Correcting Code (ECC) to mitigate SEU effects.

CMT provide the most flexible and highest-performance clock management for FPGAs. Each CMT integrates two *Digital Clock Managers* (DCM) and one PLL. A DCM is a digital device that together with the PLL is able to perform clock distribution delay compensation, clock multiplication/division, coarse-/fine-grained clock phase shifting, and input clock jitter filtering.

DSP48Es are modules containing a 25x18 multiplier (in *Xilinx Virtex 5 QV* FPGAs) or a 18x18 multipler (*Xilinx Virtex 4 QV* FPGAs), plus a 48 bits accumulator for Multiply-And-Accumulate (MAC) operations. Moreover, they contain dedicated links among them in order to create cascade connections useful to implement high-performance arithmetic and digital signal processing operations, without wasting user logic resources. However, these components are not rad-hardened by design, so designer must apply SEU mitigation techniques on them.

High-performance parallel SelectIO banks support different voltage levels (i.e., 3.3V, 2.5V, 1.8V, 1.5V, and 1.2V), and a plenty of I/O standards (i.e., single-ended, diffential, voltage-referenced), that enables a great adaptability of these FPGAs to different communication links. Each bank contains input/output registers, rad-hardened by design, that can operate in both Single Data Rate (SDR) or Double Data Rate (DDR) modes. Moreover, they contain *Per-bit deskew circuitry* allowing for programmable signal delay internal to the FPGA. Per-bit deskew flexibly provides fine-grained increments of delay to carefully produce a range of signal delays. This is especially useful for synchronizing signal edges in source-synchronous interfaces [37].

In order to perform a comparisons with other space-grade FPGA technologies (presented in Sections 3.1 and Section 3.2), Table 3.2 shows the internal resources of the two biggest FPGAs of the *Xilinx Virtex 4 QV* (i.e., *Xilinx Virtex 4-QV XQR4VLX200* [225]) and *Xilinx Virtex 5 QV* (i.e., *Xilinx Virtex 5-QV XQR5VFX130* [228]) families.

Table 3.2: Comparison of *Xilinx Virtex 4-QV XQR4VLX200* and *Xilinx Virtex 5-QV XQR5VFX130* internal resources

	XQR4VLX200	***XQR5VFX130***
User logic	178,176 (LUT4s) 178,176 (Registers)	81,920 (LUT6s) 81,920 (Registers)
Embedded RAM/FIFO	6,048 Kbits	10,728 Kbits
Embedded Flash ROM	-	-
Clock Management Tiles	6	6
Maximum User I/Os	960	836

In order to properly compare these two FPGAs, one must remember that *XQR4VLX200* FPGA is not rad-hardened by design, so all the internal resources must be protected through the SEU-

mitigation techniques presented in Section 3.4. Thus, just a portion of the internal resources can be used to provide new functionalities, while the other must be used to mitigate the particles induced effects. After this consideration, it can be noted that *XQR5VFX130* beats, in terms of amount of available resources, *XQR4VLX200* on every kind of internal resources.

As can be seen by comparing data reported in Table 3.2 and Table 3.1, space-grade *Xilinx Virtex* FPGAs provide a higher number of logic resources than OTP and Flash-based FPGAs. Moreover, space-grade *Xilinx Virtex* FPGAs allow to reach highest performances in terms of speed.
However, the main deawback of space-grade *Xilinx Virtex* FPGAs is the configuration memory. In fact, even if in *Xilinx Virtex 5 QV* FPGAs the configuration logic is rad-hardened by design, SRAM-based configuration memory is less robust against SEU-induced effects than Flash-based and antifuse-metal technologies.
A fault in the configuration memory have disruptive consequence on the FPGA-based system, since this fault can completely change the implemented architecture leading to persistent error in the system. To tolerate this kind of faults efficient mitigation techniques have been proposed in the last years. A brief overview of these techniques is reported in Section 3.4.

Since the memory configuration drawback can be efficiently solved, in the last years space-grade *Xilinx Virtex* FPGAs have been tested in different missions (e.g. *JPL MSPI MicroSAT* and *Materials International Space Station Experiment* (MISSE)). To further demonstrate the actual interest of space agency in space-grade *Xilinx Virtex* FPGAs, Figure 3.14 shows the future missions in which these FPGAs will be used.

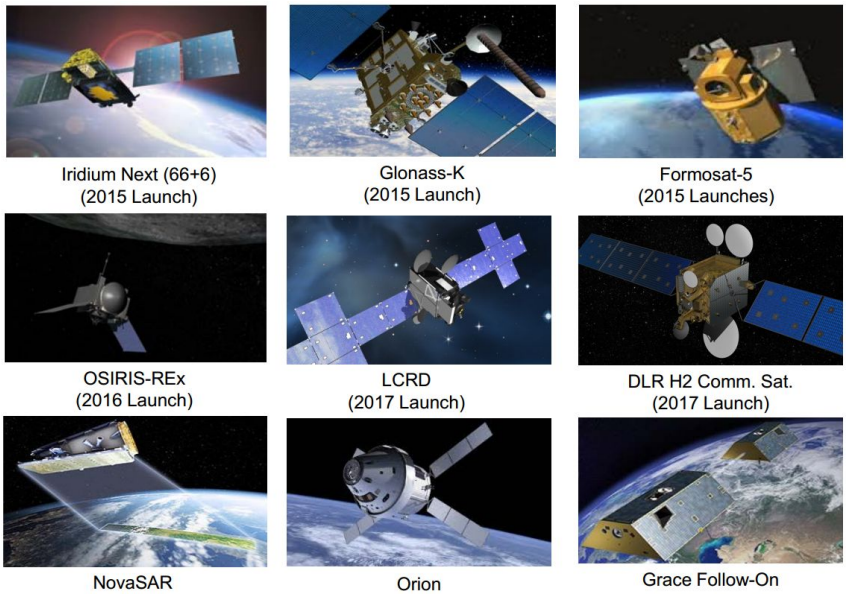


Figure 3.14: space-grade *Xilinx Virtex* FPGAs next missions

3.4 SEUs mitigations techniques

Space environment is extremely harsh for integrated circuits due to the presence of cosmic rays and alpha particlesn [109]. Phenomena caused by these elements on integrated circuits can be grouped in five categories:

- *Total Dose*
- *Dose Rate*
- *Displacement Damage*
- *Single Event Latch-up* (SEL)
- *Single Event Upset* (SEU).

Table 3.3 briefly describes effects at functional and device -level caused by these phenomena.

Table 3.3: Radiation effects description

Effect type	Functional effect	Device level effect
Total Dose	Permanent device damage due to ions over device life	Trapped positive charge results in depletion and causes leakage paths
Dose Rate	Data loss or permanent device damage due to X-rays and gamma rays	Generated photocurrent can cause transient, latch up or burnout effects
Displacement Damage	Permanent device damage due to energetic neutrons or protons	Degradation in semiconductor lattice leads to electrical parameter degradation
SEL	Instantaneous device latch-up due to single ion	PNPN path triggered into high current state at least requires power down
SEU	Instantaneous data loss due to single ion	Single memory bit or flip-flop state toggles due to ion charge

SEU is the most probable phenomena on FPGA devices. For this reason, this section is focused on the mitigation techniques against SEU induced effects.
As aforementioned, the configuration layer in OTP and Flash-based FPGAs is totally immune to SEU. Instead, in space-grade *Xilinx Virtex* FPGAs (Section 3.3) the SRAM-based switches composing the configuration layer are highly susceptible to SEU.
A SEU in an SRAM-based FPGA can flip the value of a bit associated with the configuration layer or with the user logic.

A flip in the configuration layer can have disruptive consequences, since it can change routing among logic resources or change the combinatorial function implemented in a LUT [225] [228]. Fortunately, in reality, designs only uses a small fraction of the total resources in an FPGA. No design ever uses every capability of every element in the entire device. Thus, an SEU probably will not affect the functionality of the design. Xilinx data, which has been confirmed by independent customer testing, shows that typically less than 30% of all configuration bits are used in any one design. However, even with high resource utilization, the actual rate of SEU impacting design operation is often less than 10% and is typically less than 5% [105]. This means that just few bits stored in the configuration memory can lead to a persistent error if flipped. As demonstrated in [55], *Xilinx* tools allows to easily define and extract the location of these essential bits (i.e., the configuration bits that might cause a specific customer design to malfunction). Moreover, *Xilnx* provides to designers and already implemented and validated IP-core, called *Soft Error Mitigation* (SEM) IP core [227], that allows to automatically detect and classify the SEU-induced bit-flip. If the flip happened in a essential configuration memory location, the IP-core correct the flipped bit. The IP performs the correction and classification in the background so that the system does not need to stop, increasing availability considerably.
To get a more fine grain control on the SEU mitigation process, the designer can use one of the well-known scrubbing techniques, as the one presented in [90].

Instead, a SEU affecting a bit associated with the user logic can lead to the corruption of the content of a RAM block, or the toggle of a register content. These two effects can be more or less persistent depending on the upgrading rate of the memory and registers content.

As reported in Sections 3.1, 3.2 and 3.3, the most efficient and effective way to protect BRAM blocks in every FPGA technology is the usage of an ECC associated with each word contained in the memory. For this reason for every space-grade FPGA, vendors embed or provide an IP-core to support the ECC computation.

FPGAs are devices containing a plenty of registers, so the probability that a SEU affects the content of these elements is really high. Thus, in space applications, all the registers used in the design must be protected against SEUs.
The most common technique is the *Triple Modular Redundancy* (TMR). The basic TMR approach is a hardware redundancy technique requiring the triplication of each register and then to majority vote the value of the output (Figure 3.15). This mitigation technique is able to tolerate a single fault on the redounded registers.
However, this basic approach has two main drawbacks. The former concerns the fault accumulation. In fact, if a SEU occurs on a register, the value of that register remains faulty until it is overwritten. For this reason, if the content of the faulty register is not overwritten and another of the redounded register is affected by a SEU, this approach is not able to provide in output a fault-free value. The latter concerns the single point of failure represented by the voter. In fact if a SEU corrupts the logic composing the voter, this technique provides an erroneous output value.

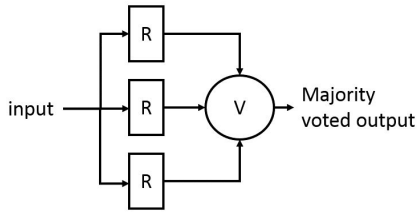


Figure 3.15: TMR basic approach

To deal with the first limitation, the TMR technique has been improved as shown in Figure 3.16.

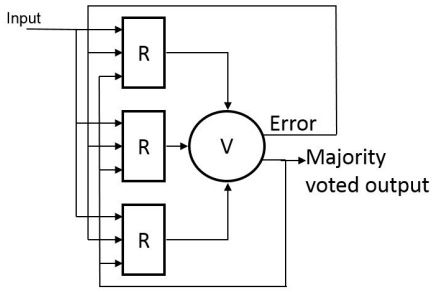


Figure 3.16: TMR improved approach

In this improved approach the majority voted output, that is assumed to be correct, is feedback to the redounded registers and the one containing a value different from the majority voted one is immediately overwritten with the fault-free value.
While the second limitation is solved by triplicating the voter as well.
The mitigation approach exploiting both of these improvements is called *Functional Triple Modular Redundancy* (FTMR) [76]. FTMR provides a highly reliable system, but it introduces an overhead of 3.2x in terms of logics resources usage and 10% timing performance degradation [71]. To reduce the introduced overhead many different TMR techniques have been proposed, starting from the ones based on time-based modular redundancy (i.e., the redundancy is achieved by repeating the execution of the operations of the module to be redounded) [126], to the one that directly operate to the internal architecture of the FPGAs [120]. The detailed analysis of these techniques is out of the scope of this section, so for further information reader may refer to the cited documents.

By analysing the internal architecture of the different space-grade technologies, it can be immediately noted that *Microsemi RTAX-S/SL* FPGA already integrates rad-hardened registers exploiting the FTMR. Thus, designers must not take care of applying mitigation techniques to the implemented design. Instead, *Microsemi RT ProAsic* and space-grade *Xilinx Virtex* FPGA con-

tain just simple registers. For this reason, designers must manually apply FTMR to every used register in order to implement a SEU tolerant architecture. Since the manual insertion of the FTMR technique can be really time consuming, in the last years vendors developed tools able to automatically apply FTMR to internal registers. In particular, the two design suites offering this feature are: *Synopsys Synplify Pro* [193], and *Xilinx TMRTool* [97]. The main difference between these two tools is that the former is FPGA vendor independent, while the second is able to deal just with *Xilinx* FPGAs. On the contrary, *Synopsys Synplify Pro* is able to apply triplication on registers, only, while *Xilinx TMRTool* triplicates the combinatorial logic, as well, to increase even more robustness against particle induced effects. For the sake of completeness, Figure 3.17 shows an example of the FTMR applied to both registers and combinatorial logic.

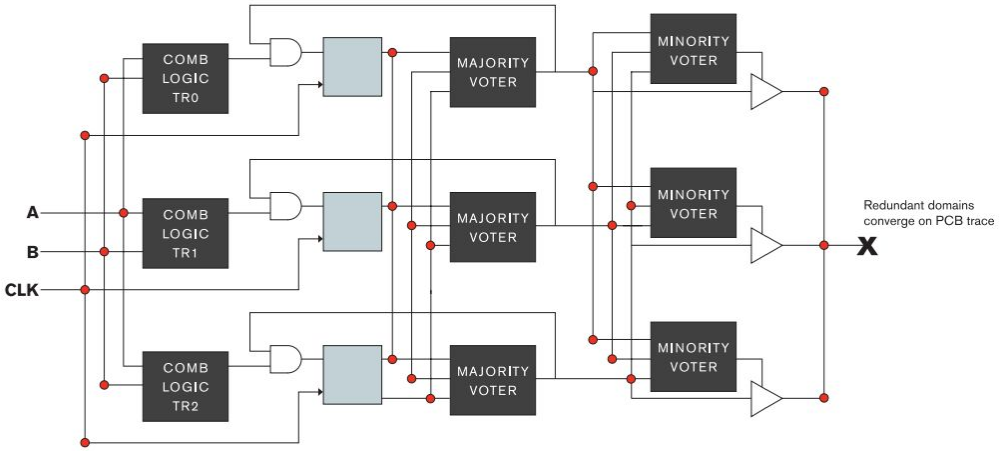


Figure 3.17: *Xilinx TMRTool* mitigation technique example [97]

A particular remark has to be performed concerning the hardening techniques for *Finite State Machine* (FSM). Since TMR at least introduces a 3x increase in resources usage, an alternative technique to hardening FSM against SEU is the ECC based technique. Unlike TMR where an FSM is simply replicated entirely, ECCs redundantly encode the FSM state variables. These ECC-based techniques can be grouped in two main categories: (i) *Explicit Error Correction*, and (ii) *Implicit Error Correction* techniques [145].

Explicit Error Correction requires to encode each state variable and add a circuity able to detect and correct error in the encoded variable. This detection and correction circuitry is added between the logic computing the next state variable value and the register storing the actual state (Figure 3.18).
Thanks to this modification to the FSM architecture, each time a SEU affects the logic computing the new state variable or the register storing the actual state, it is possible to detect the fault and avoid a wrong execution flow or to go in unexpected state.

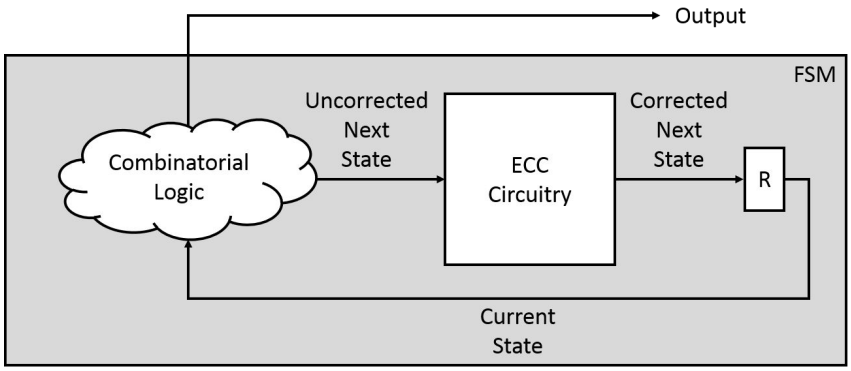


Figure 3.18: *Explicit Error Correction* hardening technique example

Implicit Error Correction, unlike explicit error correction, does not use additional hardware as error correction circuitry. Infact, to prevent that a SEU becomes a failure, the next state logic is expanded to include all the faulty state values that are a Hamming code distance of one away from the valid state value. The major advantage of this technique is that there is no need for additional error correcting logic. However, with the added states, the next state forming logic is more obtuse than in its original un-encoded form. "Don't cares" in the state logic are reduced because the set of invalid and valid codewords must be handled instead of just the valid codewords. The same principles must be applied to the output forming logic to ensure the output correctness.

CHAPTER 4

FPGA-BASED ACCELERATORS LIBRARY FOR IMAGE PROCESSING IN SPACE

According to the Moore Law, in the years the size of transistor has been shrunk more and more, allowing semiconductor industries to achieve impressive density of *Very Large-Scale Integrated* (VLSI) circuits [103]. In order to maintain under control the increased level of complexity, design engineers improved the design methodologies and techniques following the evolution of VLSI circuits. This evolution moved hardware systems from System-on-Board (SoB) to System-on-Chip (SoC). This innovation allowed to integrate in the same chip all the functionalities required by the hardware system, with an important increase in the performances and a reduction in the power consumption. However, this new methodology did not bring just benefits. In fact, with the increased integration of SoC, the test and validation of the implemented circuit has become more and more complex, time consuming, and expensive.

In order to exploit the benefit provided by SoC, while limiting the complexity of the circuit validation, an emerging methodology proposes to combine on a single chip predesigned and preverified blocks, often called *Intellectual Property* (IP) cores, or virtual components, obtained from internal sources, or third parties.
The jointly use of SoC and IP-core approaches enables a huge increase in the hardware system productivity. In fact, SoC designers, rather than design and validate each of these components, can focused their efforts in just the integration of these IP-cores in a single chip to implement complex functions, strongly reducing the design time. Thus, the design engineer is just involved in the interconnection of the IP blocks to the communication network, in the implementation of the Design For Testability (DFT) techniques [173], and in the usage of methodologies to verify and validate the overall system-level design.

In order to provide the same benefits to the space hardware engineers, this chapter presents a library of FPGA-based IP-cores for advanced image processing in space applications. It has been decided to set up this library, since in the market there are a lot of libraries containing IP-cores for

image processing [39, 98], but no one of them is tailored, in terms of algorithm self-adaptability and low hardware resources usage, to meet all the constraints required by space applications. All the IP-cores contained in the library target the FPGA technology in order to accomplish the trend of all space agencies to use this kind of devices to develop high performance hardware accelerators for space (Chapter 3).

The library has been organized in different families depending on the IP-core functionality. In particular, the contained families are:

- *Image filters*: it provides a set of IP-cores aiming at performing the image filtering task. In particular, this family is composed of two main components:
  - *2D convolver*: the main IP-core composing this family is a 2D convolver. This IP-core aims at providing a high-performance and area efficient implementation of the 2D-convolution. This operation is used to implement the most important image filters for blurring, sharpening, embossing, edge-detection, and noise reduction. In this way, with just a single IP-core it is possible to provide the designer a plenty of functionalities.
  - *Adaptive Image Denoiser IP-core* (AIDI) [54]: it is an adaptive image noise filter. This IP-core is of particular importance for space applications in which expected noise cannot be predicted at design-time. In fact, this component is able to automatically detect the level of noise affecting the input image, and boost the filtering parameters according to it, in order to achieve the best performances independently to the current image conditions.
- *Image Histogram calculator*: it contains just one IP-core that aims at computing the histogram associated with the input image. An image histogram is a graphical representation of the tonal distribution in a digital image. This graph is of particular importance in image processing, since it can be an useful tools to analyse the tonal distribution in the image to perform brightness and contrast adjustment, but, also, to define the best threshold value in edge detection, and image segmentation algorithms [82].
- *Image Enhancers*: it contains all the IP-cores aiming at enhancing the input image from the contrast point of view. This operation is of particular importance in space applications since, many times, image acquired in the harsh space environment are not well defined and they need to be enhanced to highlight their texture and information. Processing enhanced images, the performances of the following image processing steps are highly increased.
  The Image Enhancement Techniques (IET) that better improve images from the contrast point of view are those working in the intensity domain. Among these IET the ones that proved to be the most effective ones are the *Histogram Equalization* (HE) and the *Histogram Stretching* (HS) [123]. For these reasons, this library is composed of the following IP-cores:

- *Histogram Stretcher*: it accelerates the execution of the Histogram Stretching technique.
- *Histogram Enhancer*: it accelerates the execution of the Histogram Equalization technique.
- *Self-Adaptive Frame Enancher* (SAFE) [51]: it is an IP-core able to automatically select and apply, on the input image, the best IET, between HE and HS, depending on the current environmental and image conditions. The self-adaptability feature provided by this IP-core is of particular importance for space applications, in which the environmental conditions cannot be predicted a priori.

- *Features matchers*: the IP-cores contained in this library computes in an efficient way the most important area-based features matching algorithms (Section 2.5.1.2.1). These IP-cores are of particular importance for space applications, since are the most used for the features matching task in relative navigation systems (Section 2.5.1.2).
  In particular this family is composed of four IP-cores:
  - *SAD*: it accelerates the execution of the Sum of Absolute Difference (SAD) matching technique (Equation 2.10).
  - *SSD*: it accelerates the execution of the Sum of Square Difference (SSD) matching technique (Equation 2.11).
  - *CC*: it accelerates the execution of the Cross-correlation (CC) matching technique (Equation 2.12).
  - *NCC*: it accelerates the execution of the Normalized Cross Correlation (NCC) matching technique (Equation 2.13).

In order to provide a highest flexibility to the designer, all the IP-core contained in the same family are characterized by the same input/output interface. This feature ensures to easily replace, in a complex image processing chain, IP-cores of the same family without requiring modification to the communication infrastructure. Moreover, each IP-core can be customized exploiting VHDL generic [232] in order to support different image sizes and resolutions.

The proposed IP-cores provide very high performances in terms of both low resources usage and high speed. This is mainly due to a novel design methodology that allows to define a standard way to map an image processing algorithm on an FPGA device in an efficient way. This design flow supports the designer from the algorithm modelling and optimization, up to the highly optimized hardware design. The generality and flexibility of the proposed methodology makes it suitable not just for space applications, but for all of those fields requiring high performance image processors.

In addition to the design methodology, a verification and validation approach has been proposed. It allows to easily validate both the algorithm and its hardware implementation, ensuring

to discover programming bug in the algorithm model and/or design hardware errors as early as possible.

In the sequel of this chapter a detailed description of the proposed design flow and verification and validation methodologies is proposed. The last sections of this chapter deeply analyse the content of each IP-core family, highlighting the internal architecture and the performances of each IP-core.

4.1 IP-core design methodology

This section presents the proposed design methodology supporting the designer from the algorithm modelling up to the system implementation. This methodology forces the designer to apply algorithm and hardware design optimizations in the proper design flow stage, ensuring to reach highly optimized IP-cores and to speed-up the overall design process.

Figure 4.1 shows the main steps composing this methodology.

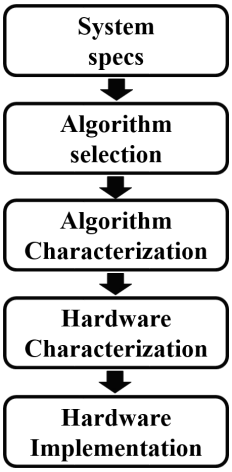


Figure 4.1: IP-cores design flow

The first step to be done is the analysis of system specifications. Commonly, designers of complex system receive as main inputs different documents that specify the requirements in terms of functionalities, accuracy of the output results, interfaces, and performances in terms of speed and area occupation. These requirements are usually embedded in the following documents: *Interface Control Documents* (ICDs), *Design Requirement Documents* (DRDs), and *Design Requirements List* (DRL). In high complex systems, the information contained in these documents are integrated with *Technical Notes* (TNO), that provide additional technical constraints concerning some peculiar parts of the system.

As shown in Figure 4.1, after the system requirements analysis, it is possible to proceed with the Algorithm Selection phase. This phase aims at comparing the requirements with the characteristics of algorithms providing the needed functionality. It starts with the study and analysis of literature to identify suitable algorithms for the target application. Then, it proceeds with the execution of different tests on the identified algorithms, exploiting algorithm models and significant image datasets (Section 4.2), and it ends with the selection of the algorithm which accuracy and complexity is closest to the system requirements.

The selected algorithm is then characterized executing the *Algorithm Characterization* task. This aims at tuning the internal algorithm parameters and, when needed, to modify parts of the algorithm, in order to meet the requirements in terms of results accuracy.
As shown in Figure 4.2, this phase requires as inputs: an algorithm model (i.e., an algorithm description in a high level programming language, e.g., C/C++ codes or MATLAB scripts), and a significant image dataset to properly characterize the selected algorithm.

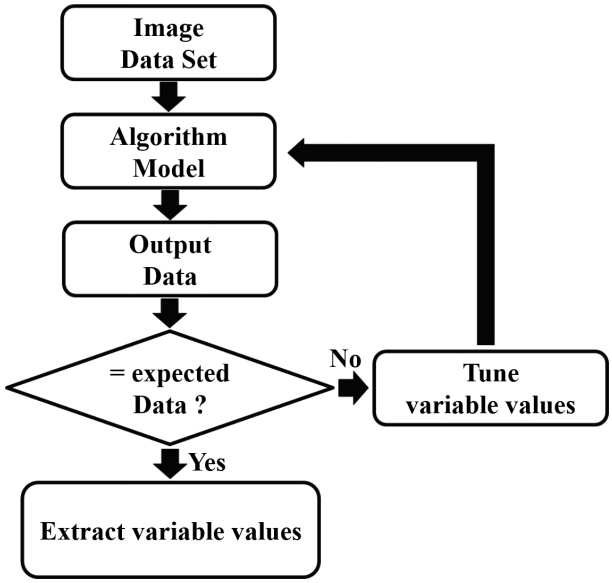


Figure 4.2: Algorithm characterization flow

To ease the selection of the most suitable image dataset, on the web a lot of freely downloadable datasets have been classified depending on the targeting image processing application (an example of classification can be found in [73]). For some applications, in order to furtherly increase the precision of the algorithm characterization, in the selected dataset are added a comprehensive set of pictures, acquired in different environmental and image conditions (i.e., different contrast, illumination, and blur conditions), representing the environment in which the algorithm has to operate.
The core operation of the *Algorithm Characterization* phase is a loops that iteratively provides in

input to the algorithm model the images composing the selected dataset, and extracts the output results. The output results are parameters that objectively quantify the accuracy of the obtained results. Some example of these parameters are the number of extracted features for a features extraction algorithm (Section 2.5.1.1), or the *Peak Signal to Noise Ratio* (PSNR) [82] for a noise filtering algorithm.
For each iteration the extracted parameters are compared with the requirements. If the obtained results are aligned with the requirements, no modification are applied to the algorithm, otherwise, the internal algorithm parameters are modified, and a new iteration started.
However, in some cases, a simple modification of the internal parameters cannot ensure to meet the required performances, especially in terms of adaptability to the environmental and image conditions. For this reasons, customization of some part of the algorithm are required to fulfil all requirements.
The outcome of this phase is a customized version of the selected algorithm that completely fits the system requirements.

Then, the customized algorithm model and the image dataset are provided in input to the *Hardware Characterization* phase. The main purpose of this phase is the definition of the data representation (i.e., data format and size) to be used in the hardware implementation.

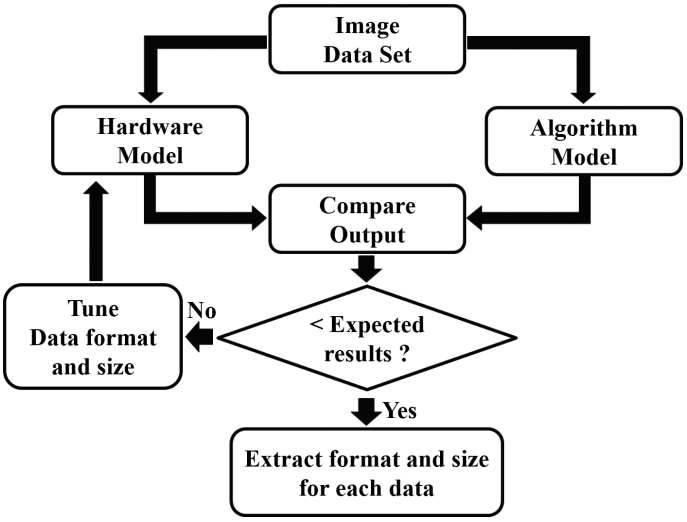


Figure 4.3: Hardware characterization flow

This is a really important phase, since it can strongly affect both the result accuracy and the hardware resources usage, especially when fractional numbers are involved in the algorithm computation.
The representation of fractional numbers in a hardware component can be obtained through either the floating point or the fixed point formats. From the one hand, the selection of floating

points representation allows to achieve really high accuracy but, at the same time, it increases the complexity, and so the hardware resources usage. From the other hand, fixed point representation enables to reduce the hardware complexity, but it does not guarantee high precision for every number range.

Since in space application high accuracy and low hardware resources usage are two key factors, the *Hardware Characterization* phase guides the designer in the selection of the data format (i.e., integer, floating point or fixed-point) and size (i.e., size of each data in terms of bit) to achieve the same accuracy of the customized algorithm model, while maintaining the hardware resources usage as low as possible.
To achieve this goal, as shown in Figure 4.3, a hardware model has to be created. This hardware model, implemented as a MATLAB script, is a high level description of the algorithm exploiting the same data representation that will be used in the hardware implementation. MATLAB scripts have been selected since, differently from others high level programming and scripting languages, provides the *Fixed Point toolbox* [134] that eases the modelling of computation in fixed-point precision.
As for the *Algorithm characterization*, an iterative optimization task is done (Figure 4.3). At the beginning, in the algorithm model, all data are represented involving the lowest precision representation. Then, for each iteration, output results obtained providing the image dataset in input to the algorithm and hardware models are compared to find differences. If no differences are found, the data representation and format adopted in the hardware model can be considered valid, otherwise the data representation is modified to slightly increase the representation accuracy, and a new iteration is done. This operation is repeated until the representation error has been confined in the desired range.

The last phase of the proposed methodology, called *Hardware implementation*, guides the designer in the mapping of the customized algorithm on an FPGA device, in order to obtain the implemented hardware component. This phase receives in input the hardware model in order to efficiently implement the customized algorithm in the FPGA device. During this implementation phase, a particular effort has to be spent on the architectural decisions to ensure the maximization of the timing performances, and so to meet the strict real-time constraint imposed by space applications.

As can be noted from the reported description, the main outcomes of the proposed methodology are the *Algorithm Model*, the *Hardware Model*, and the *Hardware Implementation*. As will be described in Section 4.2, these are three fundamental components to ease the verification and validation of the implemented IP-cores.

4.2 IP-core verification and validation methodology

The verification and validation methodology, to prove the correctness of the implemented IP-cores is herein presented. The proposed approach requires in input the three main outputs of the design methodology presented in Section 4.1: the *Algorithm Model*, the *Hardware Model*, and the *Hardware Implementation*.
As shown in Figure 4.4, the verification and validation methodology is composed of two verification steps and a final validation step.

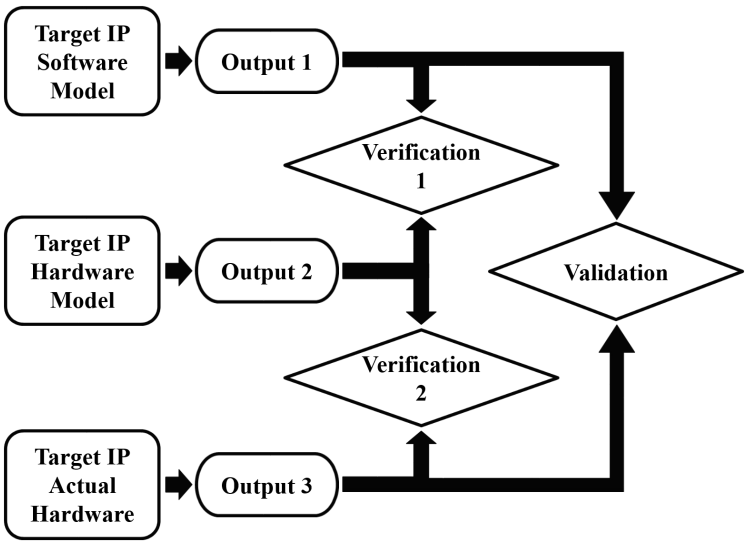


Figure 4.4: Verification and validation methodology

Verification 1 aims at verifying the correctness of the *Hardware Model*. This operation basically compares the main outputs and all the internal values of the *Hardware Model* module with the ones of the *Algorithm model*. An equivalence check is performed on the main outputs to verify the correctness of the proposed implementation, while the internal values are compared to define the error introduced by the adopted data representation.

Verification 2 aims at verifying the correctness of the *Hardware implementation*. This operation is easily done by comparing the main outputs and all the internal values of the *Hardware Model* and the *Hardware implementation*. This comparison is a simple equivalence checks, since the compared modules exploit the same internal data representation and format.

Eventually, the verification and validation process is completed by checking the equivalence among outputs of the *Algorithm Model* and the *Hardware model*. This last operation compares also the error introduced by the internal data representation and format in the *Hardware implementation* with the one computed during *Verification 1*.

To increase the effectiveness of this methodology, the algorithm and hardware models should

be described following the n-version programming paradigm [13], avoiding in this way the presence of common mode errors that could frustrate the correctness of the verification and validation task.

4.3 IP-core library

This section deeply analyses the IP-core families composing the proposed library. Since all IP-cores have been designed and validated following the methodology presented in Section 4.1 and 4.2, for the sake of brevity these processes have not been reported in this section. Instead, the description effort has been focused on the internal architecture analysis. In this way it is possible to highlight the architectural optimization that ensure to fulfil the strict requirements imposed by space applications. Moreover, at the end of the each architectural description, the achieved performances are presented.

The next subsections group the architectural description following the same IP-core classification used in the library, so each of them describes the IP-cores contained in a family.

4.3.1 Image filters

Image filters can be used for a plenty of image processing applications, spanning from the image sharpening to image deblurring, and from edge-detection to image embossing. However, the image filters family that is mandatory in every image processing chain is the *image denoising* filters.

This kind of filters allow to reduce the level of noise affecting the input image. Providing this filtered image in input to the rest image processing modules ensures more accurate results. Some examples of applications getting a benefit from this filtering task are: features identification [86], edge detection [28], and image registration [210].
One of the main reason above the importance of this filter family concerns the technology provided by modern *Charge Coupled Device* (CCD) sensors that suffers from noise. In fact, there are many potential sources of noise in a CCD camera. Dark current, Shot Noise, Read Noise and Quantization noise are just some examples [42]. CCD manufacturers typically combine these on-chip noise sources, and express them in terms of a number of electrons *Root Mean Square* (RMS) [70]. However, the level of noise in an image does not depend on the adopted sensor, only, but on the environmental condition, as well. Noise estimation and removal are thus mandatory to improve the effectiveness of subsequent image processing algorithms.

Noise sources are random in nature, their values must be handled as random variables, described by probabilistic functions, and more in particular with noise models [22]. In fact, Dark Current, proportional to the integration time and temperature, is modelled as a Gaussian distribution, Shot and Read Noise, caused by on-chip output amplifiers, are modelled as Poisson

distributions, and, detector malfunction or hot pixels are modeled by an impulsive distribution [222].
In most cases, all Gaussian and Poisson distributed noises are combined, approximating the image noise with an equivalent additive zero-mean white Gaussian noise distribution, characterized by a variance σ_n^2 [213].

As aforementioned, the noise filtering task is performed by means of a 2D-convolution with different kind of kernels [82]. However, while the impulsive noise can be removed in a relatively simple way with this approach [136], Gaussian noise removal is a non trivial task, since, to be more effective, the filter must be adapted to the actual level of noise in the image. Noise estimation is therefore a fundamental task.

For these reasons, in this IP-core family, in addition to a highly efficient 2D-convolver IP-core, *Adaptive Image Denoiser Ip-core* (AIDI) [54] has been included. AIDI is a hardware module that is able to adapt the filtering parameters depending on the level of noise affecting the input image. It first estimates the level of noise in the input image, and then applies an adaptive Gaussian smoothing filter to remove the estimated Gaussian noise. The filtering parameters are computed on-the-fly, adapting them to the level of noise of the current image. Furthermore, the filter uses local image information to discriminate whether a pixel belongs to an edge in the image or not, preserving it for subsequent edge detection or image registration algorithms.

The input and output interfaces of both IP-cores have been harmonized. These IP-cores gets the input image flow pixel-by-pixel in a raster format, line-by-line from left to right and from top to bottom, without any control or padding bit. The output is a flow in which the filtered values are provided pixel-by-pixel.

The following sections highlight the internal architecture and performances of the IP-cores contained in this family.

4.3.1.1 2D-convolver

In order to better comprehend the internal architecture of the proposed *2D-convolver*, some preliminary information about the 2D-convolution computation must be provided.
Equation 4.1 reports the formula to compute the 2D-convolution.

$$FI(x,y) = \sum_{i=0}^{N} \sum_{j=0}^{N} I\left(\delta x + i, \delta y + j\right) * K(i,j) \tag{4.1}$$

where *FI(x,y)* is the filtered pixel in position *(x,y)*, *N* is the kernel dimension, *K(i,j)* is the kernel factor in position *(i,j)* and δx and δy are described by (4.2).

$$\delta x, \delta y = x, y - \left(\frac{N-1}{2}\right) \tag{4.2}$$

In order to efficiently map these operations on FPGA hardware resources, the architecture reported in Figure 4.5 has been proposed.

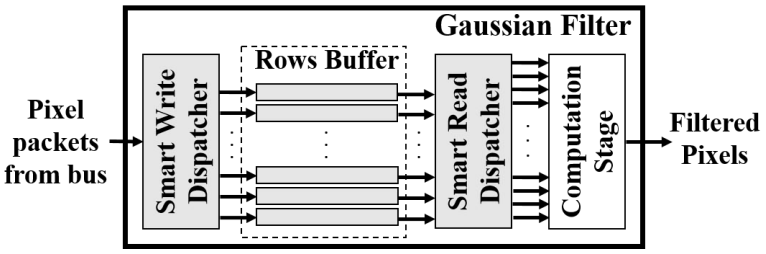
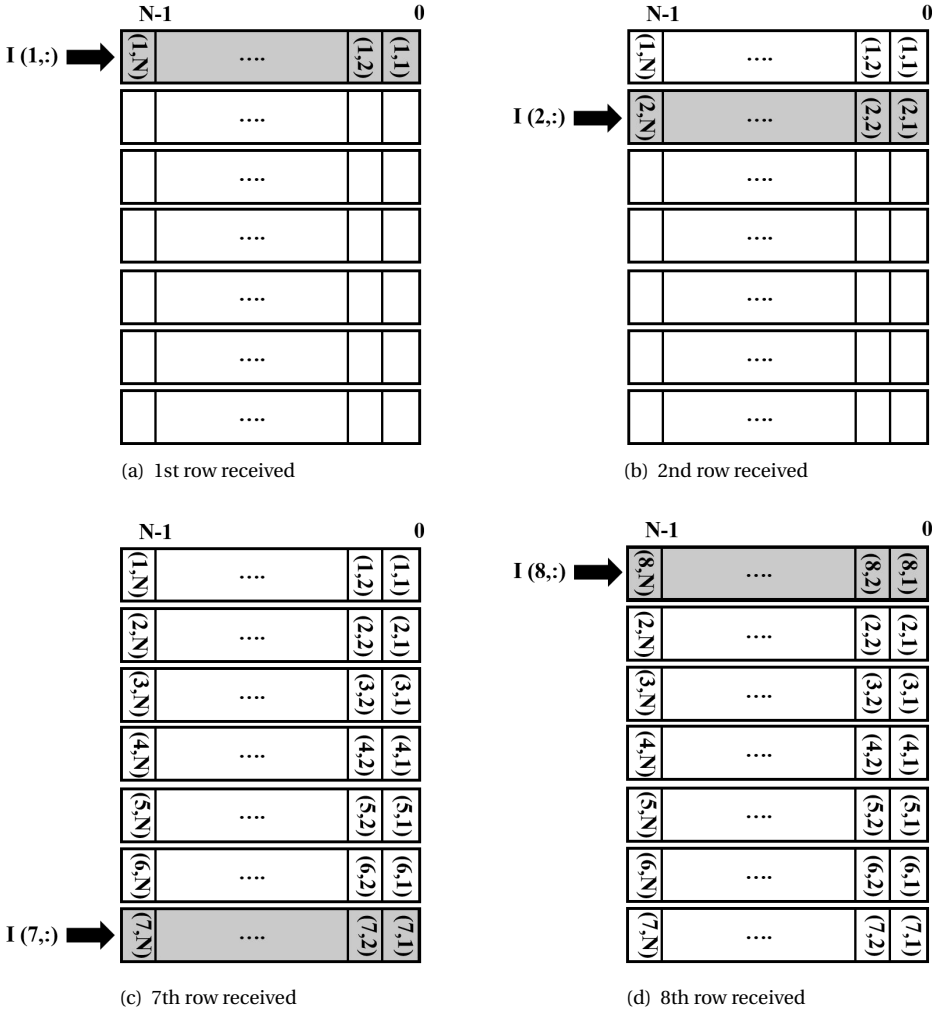


Figure 4.5: *Gaussian Filter* internal architecture

Input pixels are sent to the *Smart Write Dispatcher* (SWD), that stores them inside the *Rows Buffer* (RB) before the actual convolution computation. RB is composed of a number of Block-RAMs (BRAMs) (Section 3.3), each one able to store a full image row. The number of rows of the buffer is dictated by the number of rows of the used kernel matrix (i.e., if the kernel is 7x7, the number of BRAMs composing RB is 7). Rows are buffered into RB using a circular policy, as reported in Figure 4.6. Pixels of a row are loaded from right to left, and rows are loaded from top to bottom (from Figure 4.6(a) to Figure 4.6(c)). When the buffer is full, the following pixels are loaded starting from the first row again (Figure 4.6(d)).

The *Smart Reader Dispatcher* (SRD) works in parallel with SWD, retrieving, from RB, a set of consecutive image blocks with a size equal to the kernel one. This task is accomplished following a sliding window approach on the image (Figure 4.7).
The SRD activity starts when a number of image rows equal to the kernel row number are loaded in RB. At this stage, pixels of the central row can be processed and filtered. It is worth to remember here that, using a kernel matrix, a pixels border wide as the half of the kernel size of the image is not filtered (i.e., with a 7x7 kernel, when the kernel is centred on the pixels in the 3 pixel wide image border, some of the required pixels for the 2D-convolution computation are missing, since the kernel is partially outside the image region), and related pixels are therefore discarded during filtering. At each clock cycle, a full *RB column* is shifted into a pixels *Sliding Window Buffer* (SWB) (Figure 4.7) with the same size of the kernel. Each storage element composing this buffer is implemented with a register. After a number of clock cycles equal to the number of columns composing the kernel, the first image block is ready for convolution. The *Computation Stage* (CS) convolves it by the *Kernel Mask* and produces an output filtered pixel. At each following clock cycle, a new RB column enters the SWB and a new filtered pixel of the row is produced.

While this process is carried out, new pixels continue to feed RB through SWD, thus implementing a fully pipelined computation. When a full row has been filtered, the next row can be therefore immediately analysed. However, according to the circular buffer procedure used to fill RB, the order in which rows are stored changes. In order to understand this process in the sequel an example taking into account a 7x7 kernel is reported. Let us consider Figure 4.8, in which rows from 2 to 8 are stored in RB, with row 8 stored in the first position. Row 8 has to feed the last line



(a) 1st row received

(b) 2nd row received

(c) 7th row received

(d) 8th row received

Figure 4.6: *Smart Write Dispatcher* operations. (i,j) indicates pixel coordinates

of the SWB. To overcome this problem, the SRD includes a dynamic connection network with the SWB. This network guarantees that, while rows are loaded in RB in different positions, SWB is always fed with an ordered column of pixels.

Eventually, the *Computation Stage* performs the matrix convolution using the MUL/ADD tree architecture presented in [49]. The tree executes a set of multiplications in parallel and then adds all the results. To properly perform this parallel computation, the *Computation Stage* contains a number of multipliers equal to the number of elements composing the kernel (i.e., with a 7x7 kernel, 49 multiplier are used), and a balanced adder tree to sum these values in a pipelined manner. Adders and multipliers must support fixed-point representation, because many kernels

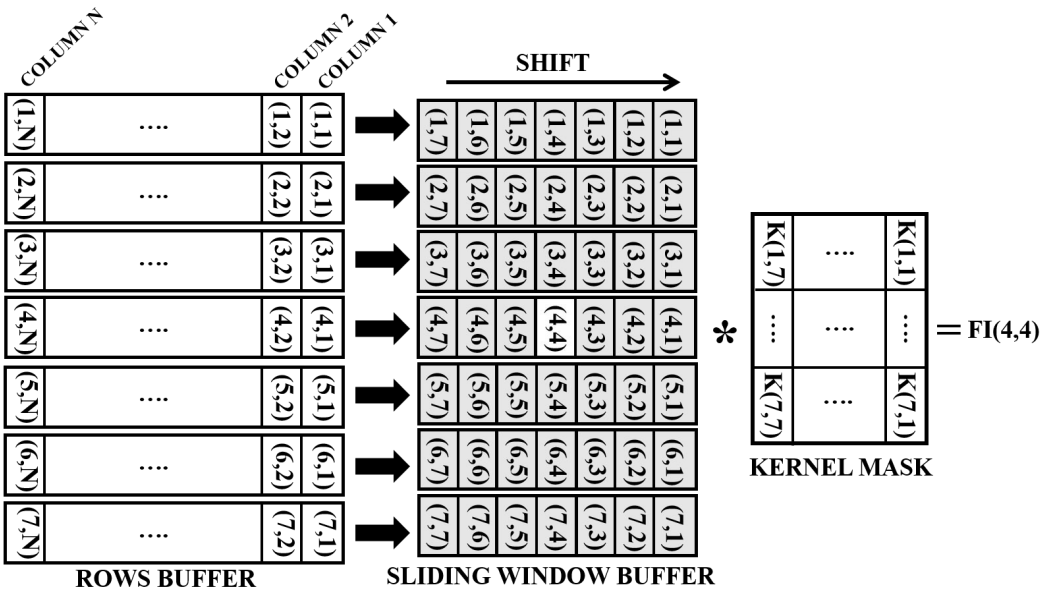


Figure 4.7: SRD behavior example. Pixel (4,4) is elaborated and filtered.

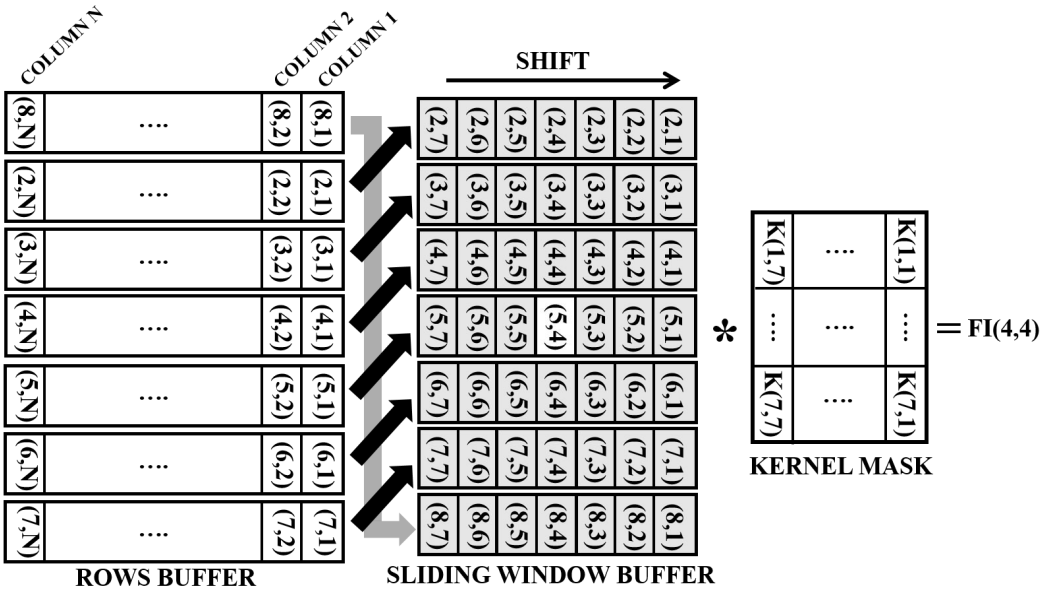


Figure 4.8: SRD behavior example. Pixel (5,4) is elaborated and filtered.

used in image processing contain fractional numbers.

The main benefits, as can be noted from the internal architecture description, are: (i) the IP-

core does not require any access to the external memory to retrieve image pixels, since all the required values are temporarily stored inside the core, and (ii) the proposed architecture allows a pipeline computation without the need to stall the input pixels flow.

For the sake of completeness, Table 4.1 shows the area occupation, the timing performance of the *2D-convolver* implemented on a space-qualified *Xilinx Virtex 4-QV XQR4VLX200* (Section 3.3). The data reported in the table are related to a configuration allowing to filter, with a 7x7 Gaussian kernel, an image composed of 1,024 x 1,024 pixels with a bpp resolution of 10 bit.

Table 4.1: Resources usage and power consumption of *FEMIP* implemented on a *Xilinx Virtex 4-QV XQR4VLX200*

FPGA Area Occupation			**Max.Freq.**
Registers	**LUTs**	**BRAMs**	**[MHz]**
696 (0.30%)	5,896 (3.31%)	7 (2.08%)	118.36

Since this core is able to operate in a fully pipelined manner, in this configuration, taking into account the maximum working frequency, the input throughput can be boosted up to 112.88 fps.

4.3.1.2 Adaptive Image Denoiser IP-core (AIDI)

AIDI is a highly parallelized IP-core able to self-adapt the Gaussian filter to the current input image.It applies the approach presented in [47], that can be mathematically formalized as follow:

$$\sigma_f^2(x,y) = \begin{cases} k \cdot \frac{\sigma_n^2}{\sigma_{OI}^2(x,y)} & \text{if } \sigma_n^2 << \sigma_{OI}^2(x,y) \\ k & \text{if } \sigma_n^2 >> \sigma_{OI}^2(x,y) \end{cases} \tag{4.3}$$

where $\sigma_f^2(x,y)$ is the variance of the Gaussian filter to be applied at the pixel of the input image in (x,y) position, σ_n^2 is the estimated white Gaussian noise variance of the input image, k is a constant equal to 1.5, and $\sigma_{OI}^2(x,y)$ is the local variance of the image without noise (i.e., noise free image) in (x,y) pixel, that can be computed as:

$$\sigma_{OI}^2(x,y) = \sigma_{NI}^2(x,y) - \sigma_n^2 \tag{4.4}$$

where $\sigma_{NI}^2(x,y)$ is the local variance associated with the noisy input pixel image.
Basically, this algorithm adapts the variance of the Gaussian filter ($\sigma_f^2(x,y)$) pixel-by-pixel, in order to strongly reduce the noise in smoothed image areas (i.e., low image local variance $\sigma_{OI}^2(x,y)$), and to reduce the distortion in areas with strong edges (i.e., high $\sigma_{OI}^2(x,y)$). In other words, $\sigma_f^2(x,y)$ is increased in the first case and decreased in the second one. $\sigma_f^2(x,y)$ can range from values near 0 to 1.5.

AIDI includes three main modules (Figure4.9): the *Local Variance Estimator* (LVE), the *Noise Variance Estimator* (NVE) and the *Adaptive Gaussian Filter*.

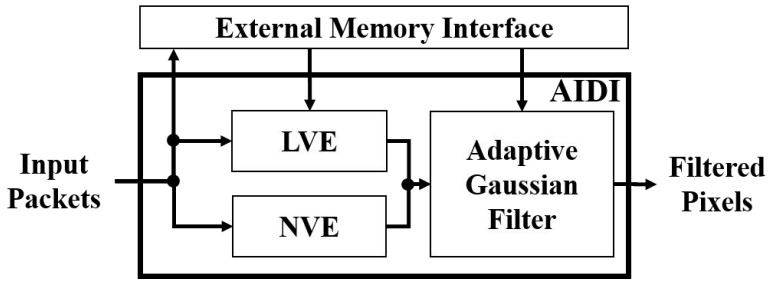


Figure 4.9: AIDI internal architecture

First, the input pixels feed the NVE and, in parallel, they are stored into an external memory.

The NVE, exploiting the algorithm presented in [196], computes the Gaussian noise variance (i.e., σ_n^2) affecting the input image. The selected algorithm involves highly parallelizable operations. It first requires to extract the strongest edges (or features) of the input image exploiting the Sobel features extractor. This task is performed using two 2D convolutions [50] between the input image and the Sobel kernels (Equation 4.5) [82].

$$G_x = I(x,y) * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, G_y = I(x,y) * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G = |G_x| + \left|G_y\right| \tag{4.5}$$

where $I(x,y)$ is the pixel intensity in the (x, y) position of the input image, and G is the edge map associated with the input image.
The strongest edges are then extracted by selecting the highest 10% values inside G. Finally, σ_n^2 can be computed as:

$$\sigma_n^2 = \left(C \cdot \sum_{I(x,y) \neq edge} \left| I(x,y) * N \right| \right)^2 \tag{4.6}$$

where N is the 3x3 Laplacian kernel [196] and C is a constant defined as:

$$C = \sqrt{\frac{\pi}{2}} \cdot \frac{1}{6(W-2)(H-2)} \tag{4.7}$$

where W and H are the width and height of the input image, respectively (in our architecture $W = H = 1024$).

When the computation of σ_n^2 is completed, the overall image is read out from the external memory and provided in input to the LVE. The LVE computes the local variance associated with each input pixel ($\sigma_{NI}^2(x,y)$). The local variance of a pixel is defined as the variance calculated on

an image window (i.e., patch) centred around the considered pixel.
To perform this task, LVE applies the following formula [192]:

$$\sigma^2_{NI}(x,y) = S - \left(\frac{1}{T}\sum_{(x,y)\in patch} I(x,y)\right)^2 \tag{4.8}$$

where T is a constant equal to the number of elements in the patch (a 11x11 pixels patch has been selected in our architecture to ensure an accurate local variance estimation), and S is equal to:

$$S = \left(\frac{1}{T}\sum_{(x,y)\in patch} I(x,y)^2\right) \tag{4.9}$$

Since LVE has a pipelined internal architecture, at each clock cycle it provides in output the $\sigma^2_{NI}(x,y)$ and the related pixel values composing the patch.

The *Adaptive Gaussian Filter* receives the σ^2_n computed by NVE, and the outputs of the LVE. The filter computes equations (4.3) and (4.4), in order to find the best filter variance value (i.e., $\sigma^2_f(x,y)$). After this computation, this module applies the Gaussian smoothing on the current received pixel.
The Gaussian filtering operation is performed by means of a 2D-convolution on the input image with a 11x11 pixels Gaussian kernel. The selected filter size allows to accurately represent the Gaussian function with variance values in the selected range (i.e., (0, 1.5], as described before). The values of the Gaussian kernel are adapted pixel-by-pixel, depending on the computed $\sigma^2_f(x,y)$, as described in Subsection 4.3.1.2.3.

In the following subsections all the hardware implementation details of the AIDI modules are deeply analysed.

4.3.1.2.1 Noise Variance Estimator

The NVE module receives the input pixels flow and it provides in output the estimated white Gaussian noise variance σ^2_n affecting the image. The internal architecture of NVE is shown in Figure 4.10.
Since NVE must perform operations involving patches the *2D-convolver*, presented in Section 4.3.1.1, is exploited. In particular, this operation is adopted for computing the convolution in Equation 4.6 and the G_x and G_y values. Since all of these operations requires the same input and they are characterized by the same kernel size (i.e., 3x3), the modules managing the input in the *2D-convolver* (i.e., *Smart Write Dispatcher*, *Row Buffer* and *Smart Read Dispatcher* in Figure 4.5) have been grouped in the SIWB module, and shared among the G_x, G_y and the *Laplacian* modules. For this reason these modules represent just the *Computational Stage*, implemented as a MUL/ADD tree, to perform the actual 2D-convolution computation. It must be precised that, since the Sobel kernel factors (Equation 4.5) can only be equal to 1, -1, 2 or -2, in order to reduce the area occupation, the multipliers are replaced by a wire, a sign inverter, a shifter, and a sign inverter & shifter, respectively.

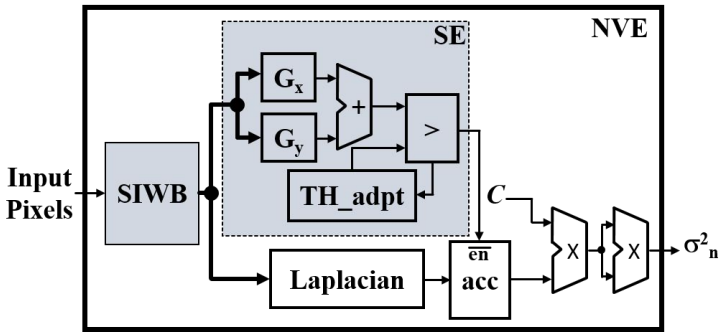


Figure 4.10: NVE internal architecture

The outputs of the G_x and G_y are then added together, through an adder, to find the G value (Equation 4.5). The computed G is compared with a threshold in order to set the SE output only if the current pixel is one of the 10% strongest features in the image.
The threshold value cannot be determined at design time since it strongly depends on the camera and environment conditions. Thus, the *TH_adpt* module (Figure 4.10) is in charge of calculating the initial threshold value and adapting it frame by frame, by simply applying Algorithm 1.

Algorithm 1 Adaptive Thresholding algorithm

$N\_target\_features \leftarrow 0.1 * size(G)$
$Gap \leftarrow N\_Sobel\_features - (N\_target\_features)$
$Offset \leftarrow Gap * (0.5/3000) * Current\_TH$
if $Gap < -3000$ || $Gap > 3000$ **then**
  $New\_TH \leftarrow Current\_TH + Offset$
else
  $New\_TH \leftarrow Current\_TH$
end if

where *N_target_features* represents the strongest features in the input image (i.e., the 10% of the complete image). *Gap* is the difference between the current number of extracted Sobel features (*N_Sobel_features*) and *N_target_features*.
If the value of *Gap* is less than -3000 or more than 3000, the current value of the threshold (i.e., *Current_TH*) is incremented or decremented (depending on its value) by one *Offset*. The new calculated value for the threshold (i.e., *New_TH*) represents the threshold to be provided in input to the comparator for the next input image.
Since at high frame rates the image conditions between two consecutive frames are approximately the same, the threshold value calculated from the previous frame can be applied to the current processed frame. This task is performed for every input frame, in order to maintain the number of extracted features around *N_target_features*. Obviously, at startup the *Current_TH* is initialized to a low value, and experiments using the hardware model of NVE, applied on the

Affine Covariant Regions Datasets [164], have shown that *TH_adpt* need a maximum of 8 frames to reach a stable threshold value.

The *Laplacian* output is provided in input to an accumulator (*acc* in Figure 4.10). This accumulator is enabled only when SE provides in output a zero, in other words only when the current processed pixel is not one of the 10% strongest features. In this way, when the complete image has been received *acc* contains the value of the sum in Equation 4.6 (i.e., $\sum_{I(x,y)\neq edge} \left| I(x,y) * N \right|$). The following two multipliers conclude the computation of Equation 4.6. To ensure a minimal error, the C constant needs to be represented in the 0.25 fixed-point format and, for the same reason, the following multipliers maintain the same number of bits for the fractional part. The estimated noise variance in output is then truncated to 12.25 fixed-point format. Thus, the NVE is able to estimate Gaussian noise variance values up to 4,000.

Finally, to improve the timing performances of the NVE module, pipeline stages have been inserted in the MUL/ADD trees and between the two output multipliers.

4.3.1.2.2 Local Variance Estimator

The LVE module receives in input the pixels read from the external memory, and it provides in output $\sigma^2_{NI}(x,y)$, computed exploiting Equation 4.8. The internal parallel architecture of LVE is shown in Figure 4.11.

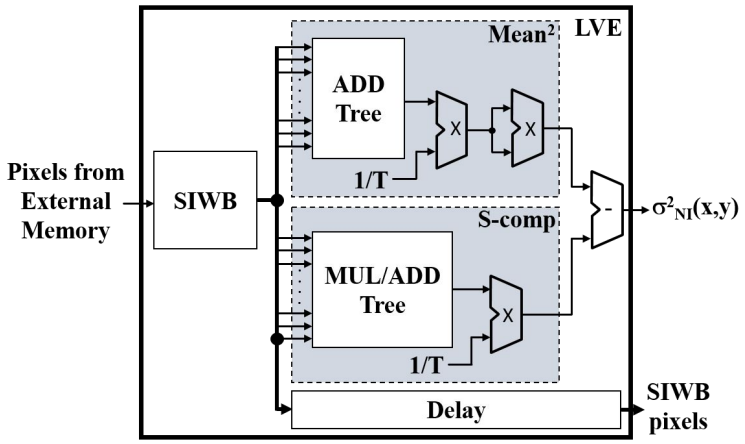


Figure 4.11: LVE internal architecture

It is composed of three main blocks: the SIWB, the *Mean*^{2} and the *S-comp*.

Since both *Mean*^{2} and *S-comp* perform operations involving patches, the input pixels are stored exploiting the same buffering approach adopted in the NVE module (i.e., SIWB explained in Section 4.3.1.2.1). The only difference concerns the *Row Buffer* (Section 4.3.1.1), which is composed of 11 BRAMs, because the LVE operations involve 11x11 pixels patches.

The SIWB output pixels are provided in input to the *Mean*^{2} and the *S-comp* modules. Moreover, the SIWB output pixels are also provided in output of LVE.

Mean^2 computes the second term of Equation 4.8 (i.e., $\left(\frac{1}{T}\sum_{(x,y)\in patch} I(x,y)\right)^2$). The received pixels are sent to the *ADD tree*, that computes the sum by means of a balanced tree composed of 7 adder stages, for a total amount of 120 adders. Finally, the output of the tree is sent to the two following multipliers to complete the computation of the second term of Equation 4.8. To ensure a high precision, the value of the $1/T$ constant and of the two multiplier outputs are represented in fixed-point format, with 15 bit for the fractional part.

In parallel to the operations performed by *Mean*^{2}, *S-comp* computes the S variable (Equation 4.8). The outputs of SIWB are provided in input to the *MUL/ADD Tree*. This tree is composed of a multiplier stage (i.e., 121 multipliers), that computes the square of the pixels in the current patch, and 7 adder stages (i.e., 120 adders), that compute the sum in Equation 4.9. In order to obtain the S value, the output of the tree is multiplied by the $1/T$ constant.

Finally, the local variance $\sigma^2_{NI}(x,y)$ is computed as the difference between the output of the *S-comp* module and the one of the *Mean*^{2} module, resorting to a subtractor.

Several pipeline stages have been inserted to improve the timing performances of the LVE module. For this reason, since $\sigma^2_{NI}(x,y)$ must be provided in output with the associated patch, the SIWB pixels are delayed in order to synchronize the LVE outputs.

4.3.1.2.3 Adaptive Gaussian Filter

The *Adaptive Gaussian Filter* receives the σ^2_n, the $\sigma^2_{NI}(x,y)$, and the pixels in output from the SIWB of the LVE (Section 4.3.1.2.2), and it outputs a filtered pixel each clock cycle. The internal architecture of this module is summarized with Figure 4.12.

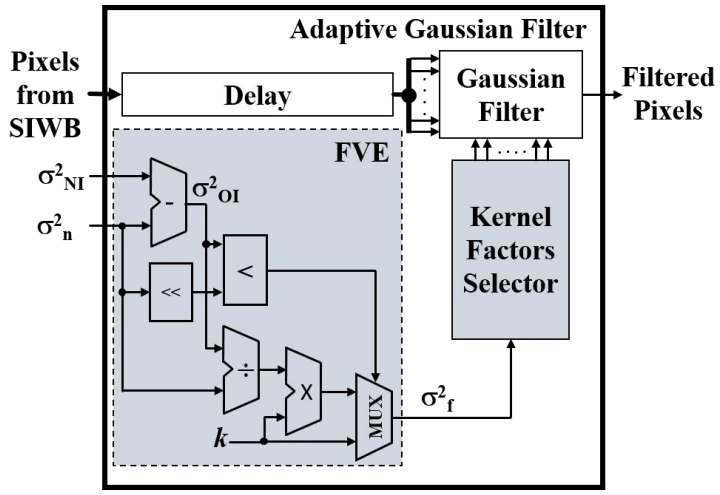


Figure 4.12: Adaptive Gaussian Filter internal architecture

The *Adaptive Gaussian Filter* is composed of three main modules: the *Filter Variance Estimator* (FVE), the *Kernel Factors Selector* (KFS), and the *Gaussian Filter*.

FVE computes σ_f^2 by applying Equation 4.3. The algorithm characterization phase (Section 4.1), performed using the *Affine Covariant Regions Datasets* [164], demonstrated that Equation 4.3 can be simplified to increase the performances exploiting Algorithm 2.

Algorithm 2 Modelled selection condition

if $\sigma_{OI}^2(x,y) < 2\sigma_n^2$ **then**
  $\sigma_f^2(x,y) \leftarrow k \cdot \frac{\sigma_n^2}{\sigma_{OI}^2(x,y)}$
else
  $\sigma_f^2(x,y) \leftarrow k$
end if

The selected model allows a very efficient hardware implementation of the selection condition by simply adopting a shifter and a comparator (Figure 4.12). Then, $\sigma_f^2(x,y)$ is computed using a pipelined divider and a multiplier and it is provided in input to KFS.

KFS aims at defining the Gaussian kernel factors associated with the current $\sigma_f^2(x,y)$. These values cannot be computed in real-time, because the associated formula [82] is very complex and time consuming, so they are precomputed and stored inside the IP.
Since each value of $\sigma_f^2(x,y)$ (represented using 31 bit) has a different associated kernel of 121 factors (i.e., the size of the kernel used to perform the filtering task is 11x11 pixels), a huge amount of data should be stored ($2^{31} \cdot 121$ kernel factors). In order to reduce the required memory resources, in the proposed hardware implementation, the range of $\sigma_f^2(x,y)$ (i.e. (0, 1.5]) has been discretized adopting a resolution of 0.1. In this way, the number of sets of 121 Gaussian kernel factors has been limited to 14. Moreover, the required storage capability has been limited exploiting the symmetry of Gaussian kernel, also. Since Gaussian kernels are circularly symmetric matrices, many factors inside them are equal to each others. Figure 4.13 shows an example of a 5x5 Gaussian kernel structure, in which the kernel factors to be stored have been highlighted.

$\mathbf{a_{00}}$	$\mathbf{a_{10}}$	$\mathbf{a_{20}}$	$\mathbf{a_{10}}$	$\mathbf{a_{00}}$
$\mathbf{a_{10}}$	$\mathbf{a_{11}}$	$\mathbf{a_{21}}$	$\mathbf{a_{11}}$	$\mathbf{a_{10}}$
$\mathbf{a_{20}}$	$\mathbf{a_{21}}$	$\mathbf{a_{22}}$	$\mathbf{a_{21}}$	$\mathbf{a_{20}}$
$\mathbf{a_{10}}$	$\mathbf{a_{11}}$	$\mathbf{a_{21}}$	$\mathbf{a_{11}}$	$\mathbf{a_{10}}$
$\mathbf{a_{00}}$	$\mathbf{a_{10}}$	$\mathbf{a_{20}}$	$\mathbf{a_{10}}$	$\mathbf{a_{00}}$

Figure 4.13: Example of a 5x5 Gaussian kernel structure

Since in a 11x11 Gaussian kernel the number of distinct kernel factors is equal to 21, in the pro-

posed hardware architecture the internally stored data for each $\sigma_f^2(x,y)$ has been limited to this value.
For these reasons, KFS has been implemented as a cluster of 14 multiplexers, in which each multiplexer is driven by the same selection signal, whose value is defined by the current $\sigma_f^2(x,y)$. In this way, the cluster of multiplexers is able to provide in output the 21 factors useful to represent the Gaussian kernel associated with the current $\sigma_f^2(x,y)$. Finally, the multiplexer outputs are duplicated in order to reconstruct the complete set of 121 kernel factors for a given $\sigma_f^2(x,y)$.
The reassembled set of kernel factors are then provided in input to the the *Gaussian Filter* together with the input pixels from the SIWB, that are delayed to be synchronized with the kernel factors.

Gaussian Filter computes the 2D convolution between the input pixel patch (i.e., Pixels from SIWB in Figure 4.13) by means of a MUL/ADD tree composed of a multiplier stage (i.e., 121 multipliers) and 7 adder stages (i.e., 120 adders).

4.3.1.2.4 AIDI performances

To evaluate the hardware resources usage and the timing performances, the proposed architecture has been synthesized, resorting to *Xilinx ISE Design Suite 14.6* [229], on a space qualified *Xilinx Virtex 5 XQR5VFX130* (Section 3.3). Post-place and route simulations have been done with *Modelsim SE 10.1c* [83].
Table 4.2 shows the resources utilization and the maximum operating frequency of each module composing AIDI.

Table 4.2: AIDI performances on a *Xilinx XQR5VFX130* FPGA device

Module	**FPGA Area Occupation**		**Max Freq.**
	LUTs	**BRAMs**	**[MHz]**
NVE	2,436 (2.97%)	3 (1.01%)	138.35
LVE	12,792 (15.62%)	11 (3.69%)	138.04
AGF	13,975 (17.06%)	- (-%)	138.04
Total	29,203 (35.65%)	14 (4.70%)	

To compare our architecture with the FPGA-based architectures for noise estimation and static Gaussian filtering presented in [124] and [108], AIDI has been also synthesized on a *Virtex II* FPGA [231].
Concerning the NVE module, it uses 3,202 LUTs and 3 BRAMs, while the real-time noise estimator presented in [124] uses 4,608 LUTs, 72 BRAMs and 24 DSP elements.
Moreover, the proposed NVE achieves higher timing performance than [124]. In fact, the architecture presented in [124] is designed for real-time processing of 720x288 pixels images at 130 fps, while our *NVE* module is able to process frames characterized by a higher resolution (i.e., up

to 1,024 x 1,024 pixels) at 136 fps.
The performances achieved by AIDI have been also compared with the architecture presented in [108]. Regarding the area occupation on a *Virtex II* FPGA device [231], the proposed architecture uses 37,695 LUTs and 24 BRAMs, whereas the FPGA-based static Gaussian filter presented in [108] uses 22,464 LUTs, 39 BRAMs and 32 DSP elements. The higher logic resource occupation (i.e., LUTs) of the proposed architecture is due to two main aspects. The former concerns the kernel used to perform the filtering task, that in AIDI is 11x11 while in [108] is 7x7 (i.e., the 7x7 kernel size does not provide high filtering performance for high level of noise). The latter regards the adaptivity provided by *AIDI*, that is not supported by [108]. Moreover, *AIDI* provides better timing performance than [108]. In fact, AIDI is able to filter 1,024 x 1,024 pixels frames achieving a maximum output frame rate of 68 fps, while [108] process 1,024 x 1,024 pixels images with a frame rate of 48 fps.

In order to evaluate the improvements provided by AIDI w.r.t. a static Gaussian filtering approach, an evaluation campaign has been performed on the image dataset reported in Figure 4.14.
On these images, different levels of white Gaussian noise have been injected, spanning from a noise variance of 100 to 4,000, exploiting the *imnoise* function provided by the *MATLAB Image Processing Toolbox* [135]. Figure 4.15 shows some examples of the injected noise on an image.
The benefits provided by the adaptivity have been quantified computing the *Mean Square Error* (MSE):

$$MSE = \frac{1}{H \cdot W} \sum_{(x,y) \in image} \left(I(x,y) - I_F(x,y) \right)^2 \tag{4.10}$$

where H and W are the height and the width of the input image, and $I(x,y)$ and $I_F(x,y)$ are the pixel intensities in the (x, y) position of the noise free and the filtered images, respectively.
Each noisy image has been filtered using: (i) a static 11x11 Gaussian filter (*Static* in Figure 4.16) with a σ_f^2 equal to k, (ii) a *MATLAB* model of *AIDI* (*Adaptive (SW)* in Figure 4.16), involving the double precision, and (iii) the *AIDI* hardware implementation (*Adaptive (HW)* in Figure 4.16), which involves fixed-point representation. The graphs in Figure 4.16 plot the trends of the MSEs, computed for each image composing the adopted image dataset (Figure 4.14), versus the variance of the injected noise. Figure 4.16 highlights two main aspects:

1. the error introduced by the fixed-point representation w.r.t. the double precision implementation can be neglected (*Adaptive (SW)* vs. *Adaptive (HW)* in Figure 4.16);

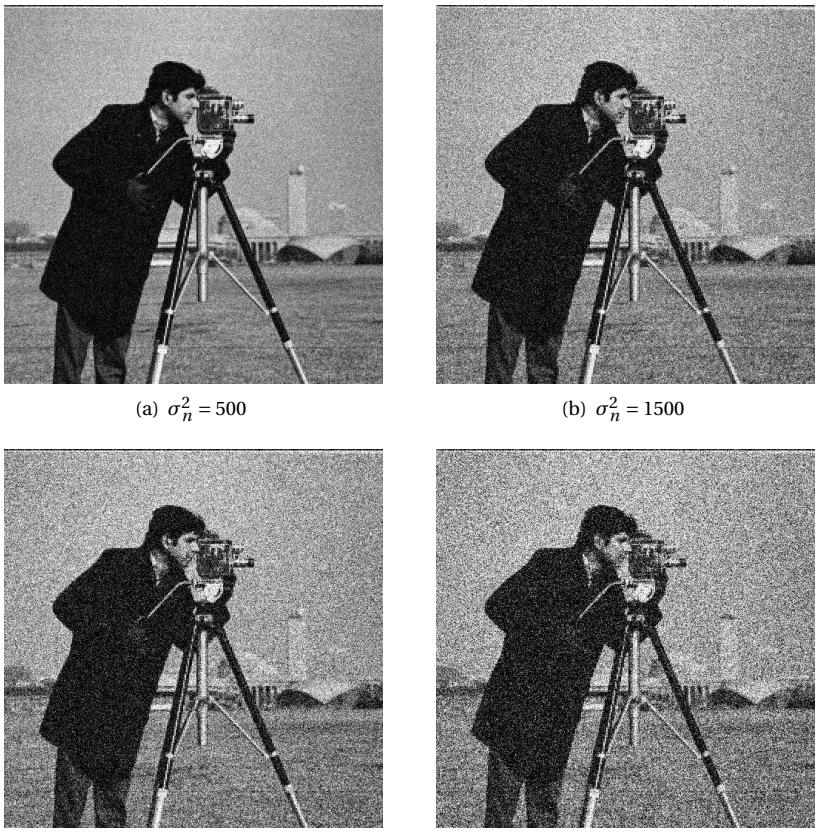
2. the MSE associated with the output of *AIDI* is always lower than the one affecting the output of a static Gaussian filter (*Adaptive (HW)* vs. *Static* in Figure 4.16). Moreover, the benefits increase for noise levels with $\sigma_n^2 \leq 1,000$, while for higher noise levels, the improvement decreases because the local variance of the image is greatly influenced by the noise, and so it cannot be accurately computed.

(a) Lena (b) Cameraman

(c) Mandrill (d) Mars

Figure 4.14: Image dataset exploited for the evaluation campaign

Finally, to prove the effectiveness of the proposed FPGA-based adaptive filter in preserving edges w.r.t. a standard static Gaussian filtering approach, the images filtered with both methods have been provided in input to a Laplacian edge detector. Figure 4.17(a) shows an example of image affected by white Gaussian noise with $\sigma_n^2 = 1,500$, while Figure 4.17(b), Figure 4.17(c), and Figure 4.17(d) show the edges extracted from the non-filtered image, the filtered image with a static Gaussian filter, and the image filtered with *AIDI*, respectively.
Despite the high injected noise, *AIDI* is able to filter the image without smoothing edges, improving the performance of the edge detector. Instead, the static Gaussian filter outputs a smoothed image, in which edges are weakened and difficult to be detected.



(a) $\sigma_n^2 = 500$ (b) $\sigma_n^2 = 1500$

(c) $\sigma_n^2 = 2500$ (d) $\sigma_n^2 = 4000$

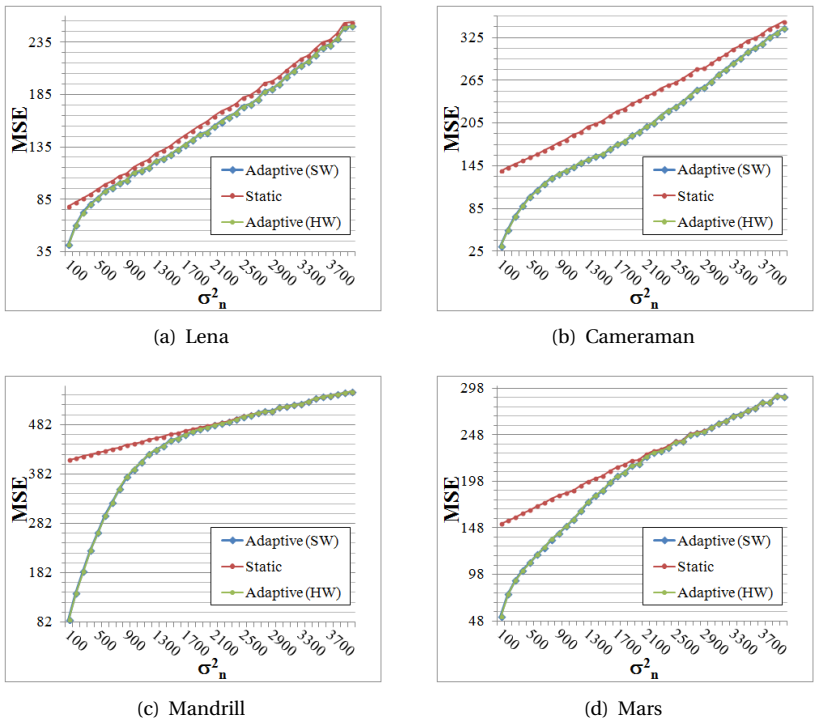
Figure 4.15: Examples of injected level of noise

4.3.2 Histogram Calculator

As aforementioned, the histogram calculation is an important task for a plenty of image processing operations. Thus, an efficient implementation of this operation enables a strong acceleration of many image processing operations.

An image histogram, in accordance with its definition reported at the beginning of this chapter, is a graph that counts the number of pixels in the image (vertical axis) that present a particular intensity value (horizontal axis). Since the available tonal value in an image is defined by the bpp resolution, the value on the horizontal axis are in the $[0, 2^{bpp}]$ range. Instead, the vertical axis can contains values up to the number of pixels composing the input image (i.e., all the pixels in the input image have the same value).

This family contains just one IP-core, the *Histogram Calculator*. This IP-core is equipped with an interface able to provide in input four pixel values in parallel at the same time. This input parallelism allows to speed up the computation task.



(a) Lena (b) Cameraman

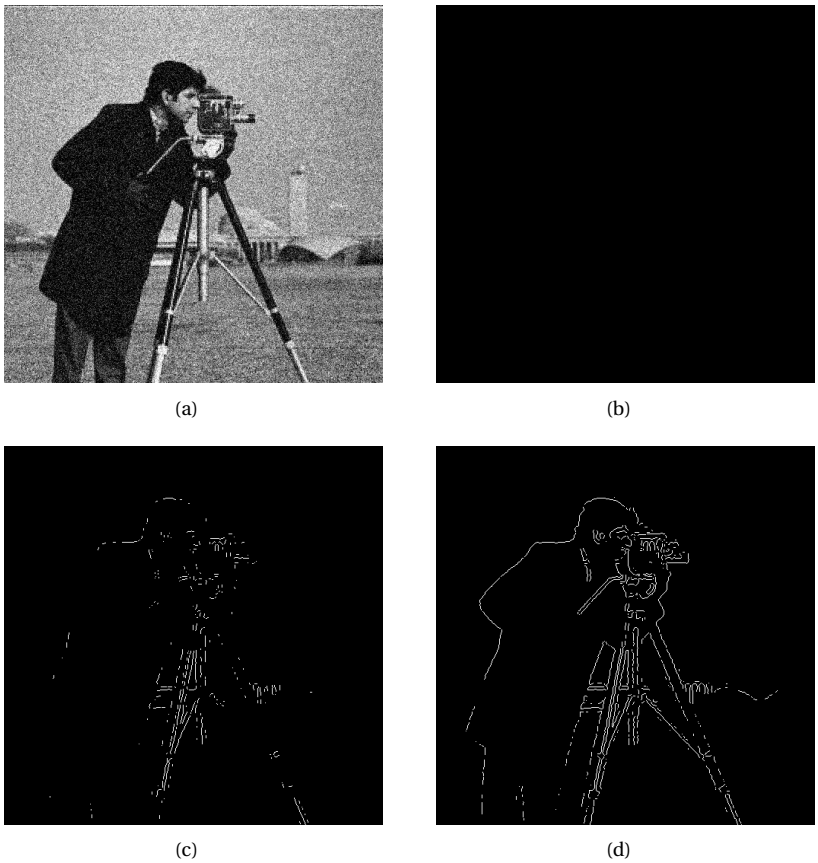
(c) Mandrill (d) Mars

Figure 4.16: Mean Square Error

The provided outputs are the values associated with the Histogram Bars (HBs), that can be internally or externally buffered. For this reason, the output interface, in addition to the HB values, provides the memory address where each HB value must be stored. Moreover, to ensure the synchronization with other modules, one additional output bit is provided to communicate the end of the histogram computation.
Thanks to this output interface, this module can be easily interconnected to a BRAM input port, for an internal buffering, or to a *Direct Memory Access* (DMA) module to store the computed histogram in an external memory.

The parallel internal architecture of the *Histogram Calculator* is shown in Figure 4.18.
It is composed of 4 *BRAM Buffers*, 4 2-inputs adders, a 4-inputs adder, 4 2-to-1 multiplexers, 8 2-to-1 multiplexers and a *Controller* that manages the overall histogram computation process.

The histogram computation is performed in two steps. First, the *Controller* sets to zero the *reset* signal. In this way input pixels act as addressing signal for the 4 *BRAM Buffers*. Buffers are implemented as dual-port Block RAMs, provided by Xilinx FPGA Virtex architectures (Section 3.3). Each BRAM Buffer has a dimension to allow the storage of all bars composing the histogram. Thus, the size of these buffers depends on the bpp resolution of the input image. Since a BRAM in space-grade Xilinx FPGAs is composed with 512 words of 32 bit each. A single BRAM buffer is



(a) (b)

(c) (d)

Figure 4.17: Laplacian Edge Extraction - (a) Noisy image in input ($\sigma_n^2 = 1500$) (b) Edges extracted from noisy image (c) Edges extracted from the image filtered by a static 11x11 filter (d) Edges extracted from image filtered by AIDI

able to support an input image with a bpp resolution up to 9 bit (i.e., the histogram is composed of $2^9 = 512$ bars). With higher resolution each buffer will be composed of more than one BRAM. The input packets are split into 4 words, each representing a pixel value. Each received pixel addresses the *BRAM Buffer* associated with its position in the input packet (e.g., the pixel in the least significant part of the input packet addresses the *BRAM Buffer 0*). The value of the location addressed by the input pixel value is read, incremented, and then rewritten in the same location in a single clock cycle exploiting the dual-port nature of the buffer. In this way, each buffer row acts as a counter. When an entire image is received, each buffer row contains a partial HB value.

In the second computation step, the partial HBs are merged to compute the image histogram. *BRAM Buffers* are scanned starting from location 0. At each clock cycle, the 4 partial HB values are read and summed. In this way, the final value of each HB is computed. After each location has been read, it is forced to 0. This ensures that counters are reset to the initial conditions, allowing

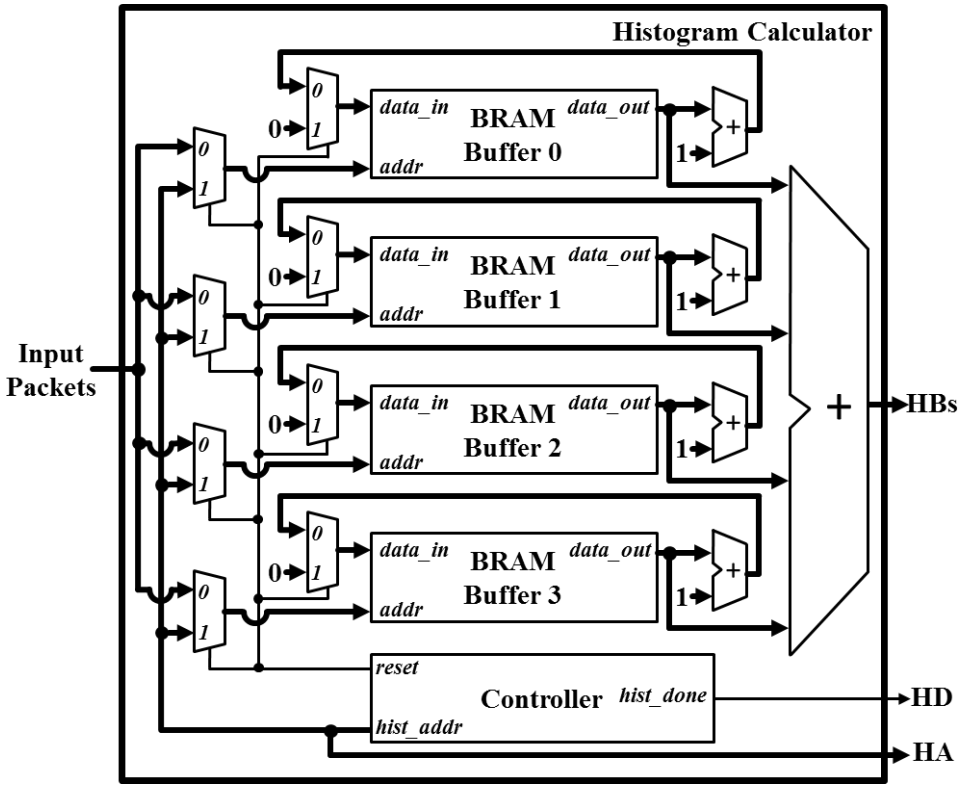


Figure 4.18: Histogram Calculator internal architecture

the computation of a new histogram.
In addition to HB values, the *Histogram Calculator* outputs the *HA* and the *HD* signals. The former represents the index associated with the output HB value, while the latter is asserted when the histogram calculation task is completed. It must be mentioned that, in the case in which the histogram must be stored in an external memory, to guarantee a proper storage the memory base address must be added to *HA*.

The first computation step performed by the *Histogram Calculator* requires the number of clock cycles needed to receive the input image, while the second step requires only 2^{bpp} clock cycles to read the buffers. Thus, the total number of clock cycles required by the *Histogram Calculator* for computing the image histogram counts up to:

$$N_{clock} = \frac{N_{row} * N_{columns}}{4} + 2^{bpp} \tag{4.11}$$

where N_{rows} and $N_{columns}$ are the number of rows and columns composing the input image.

For the sake of completeness, Table 4.3 shows the area occupation and the timing performance of the *Histogram Calculator* implemented on space-qualified *Xilinx Virtex 5-QV XQR5VFX130* (Section 3.3). The data reported in the table are related to a configuration that allows to compute

the histogram of an image composed of 1,024 x 1,024 pixels with a bpp resolution of 8 bit.

Table 4.3: *Histogram Calculator* performances on a *Xilinx Virtex 5-QV XQR5VFX130*

FPGA Area Occupation			**Max Freq.**
Slices	**DSP**	**BRAMs**	**[MHz]**
265 (0.20%)	- (-)	4 (0.67%)	74.29

The total number of clock cycles to compute the image histogram in this configuration is 262,400. Thus, the maximum sustainable throughput is 283.29 fps.

4.3.3 Image Enhancers

Image enhancement can be performed in *intensity*, *spatial* or *frequency* domains. Among the available techniques, the ones that better improve images from the contrast point of view are those working in the intensity domain. *Histogram Equalization* and *Histogram Stretching* [123] proved to be two of the most effective Image Enhancement Techniques (IET).

Histogram Equalization (HE), in particular Linear HE, changes the intensity value of each pixel to produce a new image with a more uniform image histogram (i.e., the image covers most of the brightness dynamic range). A better distributed histogram increases the image contrast, especially if the original image has close intensity values. The method is useful in images with backgrounds and foregrounds that are both bright or both dark, since these images are characterized by narrow and smoothed histograms.

The Histogram Stretching (HS) technique is based on redistribution of the pixel intensities to spread their values on the entire spectrum of colors. It increases the contrast among pixels, but it becomes ineffective when the input image features a wide histogram.

Both HE and HS are based on the statistics of the entire image, thus, after the image is analyzed, a complete buffering of the image is needed to apply the transformation, leading to a significant increase of memory occupation and latency.

Many FPGA implementations of image enhancement algorithms have been proposed: [190] [9] [117] [72]. In [190] brightness control, contrast adjustment and histogram equalization FPGA implementations are presented. Low area occupation is shown but the complete image (100 x 100 pixels) has to be internally stored.

In [9] a very efficient real-time HE technique is presented. The smart transformation function is implemented thanks to 256 16-bit counters and a hierarchical decoder. This approach clearly increases the resource usage. Furthermore, the complete image (256 x 256 pixels) is stored in a 64 kb ROM in the FPGA device.

[106] proposes a parallel HE technique. To speed up the histogram evaluation, the differential histogram calculation is used.

In [129] a simple contrast enhancement scheme is proposed, named AIVHE, for real-time image processing. The gradual increment of the brightness controls the rate of contrast enhancement by dividing the histogram in sectors.
[117] presents an Adaptive Histogram Equalization (AHE) algorithm: a method for the local contrast enhancement based on bilinear interpolation. The area occupation of the proposed AHE is considerable and the noise in background homogeneous regions is amplified.
[72] presents an implementation of Contrast Limited Adaptive Histogram Equalization (CLAHE) that limits the noise increment provided by AHE. The drawback is that a great amount of internal memory is required to store data for the bilinear interpolation.

All of the aforementioned IP-cores do not provide self-adaptive features. To overcome this limitation, in this family, in addition to two basic HE and HS hardware implementations, it has been added an image enhancer able to automatically select the best enhancement technique (i.e., HE or HS) to be applied depending on the current image and environmental conditions, called SAFE.

Since both the HS and HE require to receive in input the image histogram, the IP-cores contained in this family have an input interface, that can be directly connected to the *Histogram Calculator* outputs (Section 4.3.2). In addition to these inputs, two additional inputs have been added: BW and HW. These inputs, as described in Section 4.3.3.3, identify user-defined thresholds used by SAFE to select the best image enhancement technique to be applied. Even if they are just used by SAFE, in order to guarantee the interchangeability of IP-cores of the same family, these inputs are present in the *Histogram Stretcher* and *Histogram Enhancer* IP-cores as well, but they are left unused.

A common, really important, features of these IP-cores concerns the statistic on which the enhancement techniques are based on. In fact, they avoids the internal buffering of the image, because the enhancement is performed on the basis of tone statistics gathered from the previous image (i.e., histogram associated with the previous processed frame). This approach introduces a negligible error thanks to the limited differences between two consecutive frames, guaranteed by the high frame rate reachable by these IP-cores.

In the following sections a detailed description of the internal architecture and performances of each IP-cores is provided.

4.3.3.1 Histogram Stretcher

The Histogram stretching transformation function is reported in (4.12).

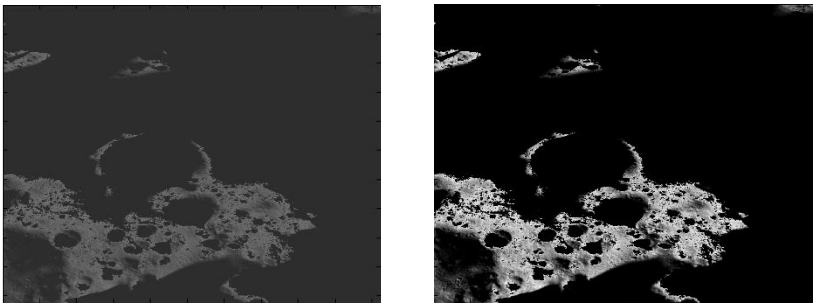
$$I_{Stretched}(x,y) = C \cdot (I(x,y) - min\_HB) \tag{4.12}$$

where $I_{Stretched}(x,y)$ is the stretched pixel intensity in the (x,y) position, $I(x,y)$ is the pixel intensity in the (x,y) position, min_HB is the minimum intensity in the previous image histogram,

and C is a scale factor equal to:

$$C = \frac{2^{bpp} - 1}{max\_HB - min\_HB} \tag{4.13}$$

In (4.13) max_HB is the maximum intensity value in the previous image histogram and bpp is the pixel resolution. This transformation of the image histogram ensures to spread the tonal distribution of the input image to cover the complete tonal range made available from the image bpp resolution. As demonstrated in Figure 4.19, this improved distribution strongly enhances the contrast of the input image.



(a) Input image (b) Stretched image

Figure 4.19: Histogram stretching example

Figure 4.20 reports the internal architecture of the Histogram Stretcher IP-core.
This IP-core is internally composed of three main modules: *Histogram Buffer*, *Histogram Stretcher Controller*, and *Stretcher*.

The *Histogram Buffer* is implemented using a BRAM, required to store the complete histogram. Each HB received from the *Histogram Calculator* (Section 4.3.2) is stored into a BRAM row. During this phase, *HD* is 0, thus *HA* acts as an address signal for the *data_in* port.
When the entire histogram is stored, the *HD* input signal is asserted and the histogram analysis, to compute the histogram width computation (i.e., the definition of the minimum and maximum intensity value in the input image, i.e., *min* and *max* in Figure 4.20), can start.

The *HW Stretcher Controller* performs two tasks: it finds the minimum and the maximum intensity values in the image histogram and it computes the histogram width. First, it scans the *Histogram Buffer* to find the first non-zero value, that represents the minimum intensity value. Then, the same operation is repeated in the reverse scanning order, to find the maximum intensity value. When both the minimum and maximum intensity value have been found, the histogram analysis is finished.
Since the operation to find these two values in the worst case requires a number of clock cycles equal to the number of bars composing the histogram, to ensure the synchronization with the

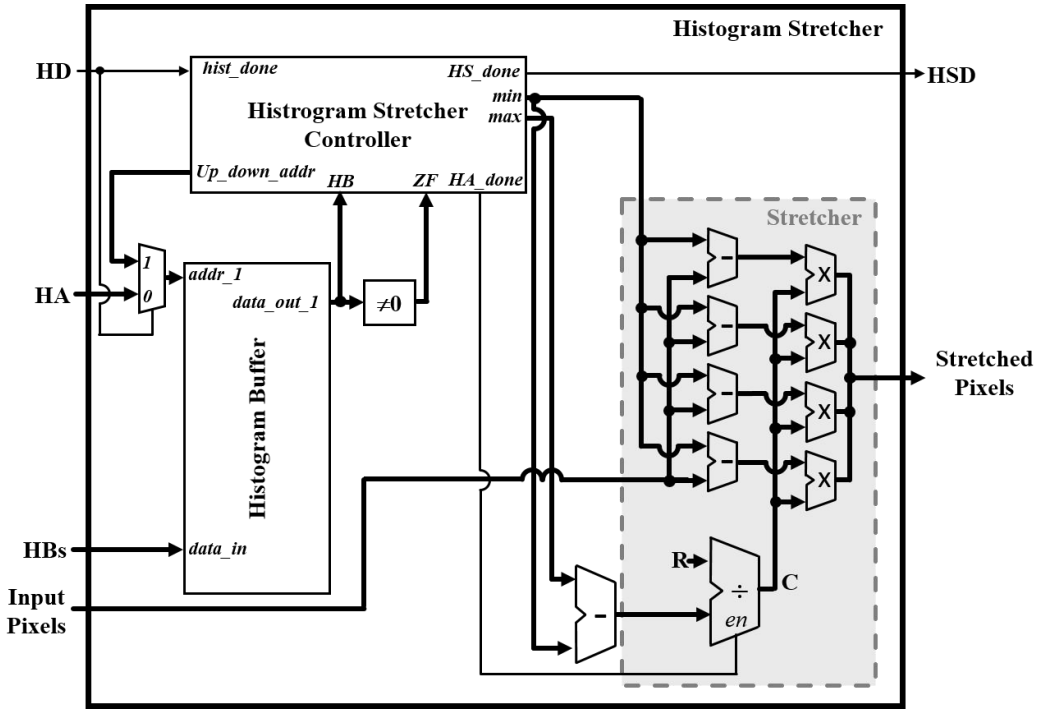


Figure 4.20: Histrogram Stretcher internal architecture

input signals, the *HA_done* signal is always asserted after this amount of clock cycles.
Minimum and maximum values are eventually subtracted to calculate the histogram width (i.e., *real_HW*).

Stretcher performs the operation shown in Equation 4.12. First, it computes the subtraction between the received pixel values and the *min_HB* signal. In parallel, the *C* factor is computed. To ensure a high precision in the division between $2^{bpp}-1$ (i.e., *R* in Figure 4.27) and *real_HW* signal, the input signals of the divider are represented in the fixed-point format with 15 bit for the fractional part. Instead, the integer part of these numbers is represented by a number of bit equal to the bpp resolution.
Finally, the results of the subtractions and the *C* factor are multiplied exploiting 4 multipliers working in parallel. The result of each multiplication is truncated in order to extract the integer part that represents the value of the stretched pixel.
When the complete image is received and, so stretched, the *HS_done* is asserted, to communicate that the histogram stretching task has been successfully computed.

For the sake of completeness, Table 4.4 shows the area occupation and the timing performance of the *Histogram Calculator* implemented on space-qualified *Xilinx Virtex 5-QV XQR5VFX130* (Section 3.3). The data reported in the table are related to a configuration that allows to stretch the histogram of an image composed of 1,024 x 1,024 pixels with a bpp resolution of 8 bit.

Table 4.4: *Histogram Stretcher* performances on a *Xilinx Virtex 5-QV XQR5VFX130*

FPGA Area Occupation			**Max Freq.**
LUTs	**DSP**	**BRAMs**	**[MHz]**
410 (2.00%)	4 (1.25%)	2 (0.67%)	72.69

4.3.3.2 Histogram Equalizer

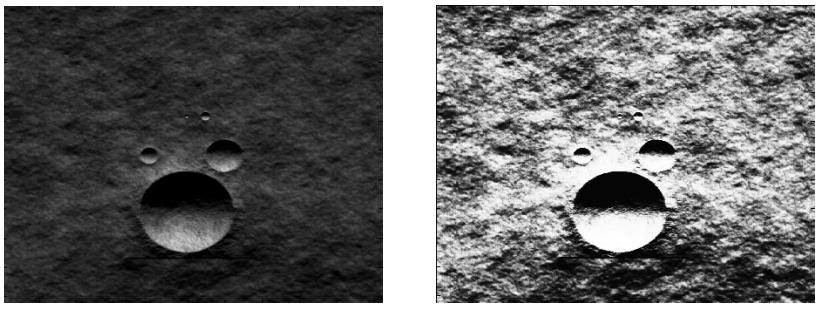
The transformation function performed during HE is:

$$I_{Equalized}(x,y) = k \cdot \sum_{j=0}^{I(x,y)} HB_j \tag{4.14}$$

where $I_{Equalized}(x,y)$ is the equalized pixel intensity in the (x,y) position, HB_j is the value of the j-th HB and k is equal to:

$$k = \frac{2^{bpp}-1}{N_{row} * N_{col}} \tag{4.15}$$

This transformation allows to increase the occurrence of the low frequency intensity values, producing in this way a highly contrasted image. To better understand this transformation, Figure 4.21 shows an example of histogram equalization.



(a) Input image (b) Equalized image

Figure 4.21: Histogram equalization example

Figure 4.22 reports the internal architecture of the Histogram Equalizer IP-core.
This IP-core is internally composed of three main modules: *Histogram Buffer*, *Histogram Equalizer Controller*, and *Equalizer*.

Until the histogram is not completely received, this module acts as the *Histogram Stretcher* (Section 4.3.3.1).
When the histogram has been completely received, and so stored in the *Histogram Buffer*, the HD

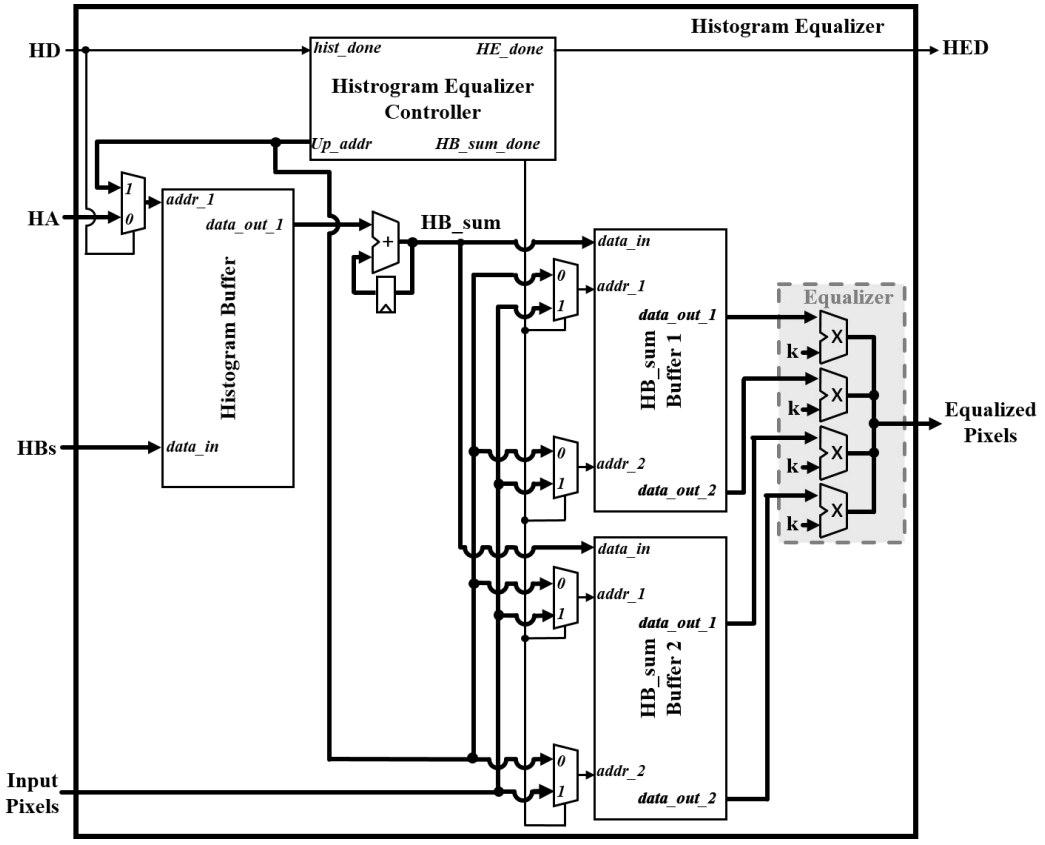


Figure 4.22: Histogram Equalizer internal architecture

signal is asserted, and the histogram analysis phase starts.
In this phase, *Histogram Equalizer Controller* scans the *Histogram Buffer* in order to compute the *HB_sum*, that represents the value of the sum of the equalization transformation function (i.e., $\sum_{j=0}^{I(x,y)} HB_j$ in Equation 4.14).
In order to perform these tasks, an accumulator is exploited. To ensure a proper computation, the size of this component is equal to the number of bit required for representing the total number of pixels composing the image (i.e., the last *HB_sum* is equal to the number of pixel composing the image). Four copies of each *HB_sum* are stored in *HB_sum Buffer 1* and *HB_sum Buffer 2*.
These copies are finally used by the *Equalizer* to apply the transformation function shown in Equation 4.14 on each input. The 4 input pixels address the 4 ports provided by the two buffers in order to read out the associated sum value (i.e., $\sum_{j=0}^{I(x,y)} HB_j$ in Equation 4.14. The 4 values are then multiplied by the k constant (Equation 4.14). To ensures a very high precision, the k constant is represented using the 0.15 fixed-point format.

For the sake of completeness, Table 4.5 shows the area occupation and the timing perfor-

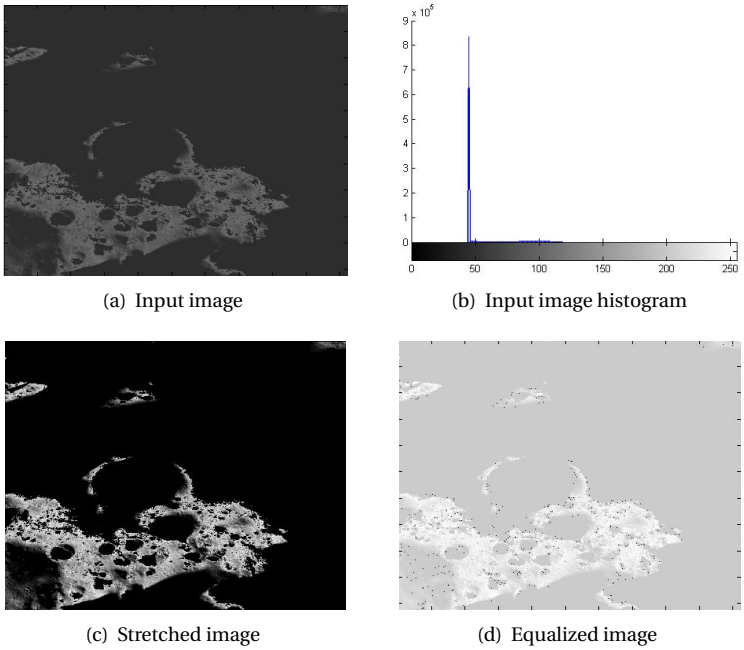
mance of the *Histogram Equalizer* implemented on space-qualified *Xilinx Virtex 5-QV XQR5VFX130* (Section 3.3). The data reported in the table are related to a configuration that allows to enhance the histogram of an image composed of 1,024 x 1,024 pixels with a bpp resolution of 8 bit.

Table 4.5: *Histogram Equalizer* performances on a *Xilinx Virtex 5-QV XQR5VFX130*

FPGA Area Occupation			**Max Freq.**
Slices	**DSP**	**BRAMs**	**[MHz]**
180 (0.88%)	4 (1.25%)	6 (2.01%)	72.69

4.3.3.3 Self-Adaptive Frame Enhancer (SAFE)

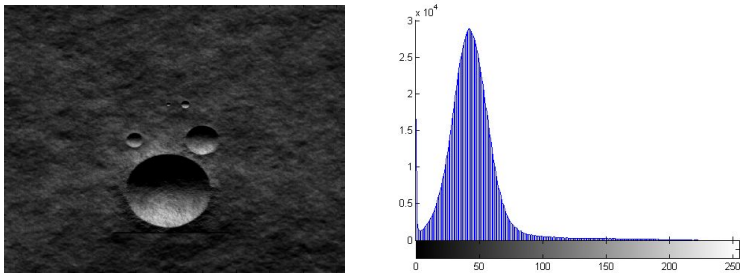
The HE and HS are really efficient techniques, but this efficiency is not valid for every kind of image. In fact, as reported in Section 4.3.3, HE works better than HE on images with a narrow and picked histogram. Instead, HE provides better performances in all other conditions (i.e., smoothed or wide histogram). These last two sentences are graphically justified in Figure 4.23 and Figure 4.24, respectively.



(a) Input image (b) Input image histogram

(c) Stretched image (d) Equalized image

Figure 4.23: Enhancement of a picture with a narrow and picked histogram



(a) Input image (b) Input image histogram

(c) Stretched image (d) Equalized image

Figure 4.24: Enhancement of a picture with a wide and smoothed histogram

In order to cope with this issue, SAFE has been proposed. It is a highly parallelized FPGA-based IP core able to select and apply the best image enhancement technique (i.e., HE or HS) depending on the input image statistics.
SAFE is composed of two main blocks (Figure 4.25): the *Histogram Analyzer* and the *Equalizer / Stretcher*.

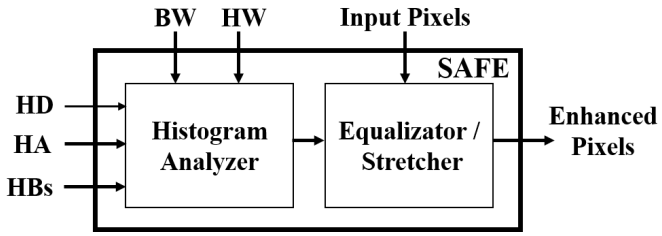


Figure 4.25: SAFE internal architecture

The *Histogram Analyzer* analyses the image histogram in order to select the best IET to be applied. It scans the histogram to find the minimum and maximum intensities and the maximum difference between two consecutive bar values. By comparing this two quantities with the HW and BW thresholds, it selects the best IET (Algorithm 3).

The *Equalizer / Stretcher* performs both HE and HS on the input image, but it provides in

Algorithm 3 Histogram Analyzer operations

```
found ← FALSE
max_BW ← 0
previous_HB ← HB(0)
for i = 0 → 255 do
    if (HB(i) ≠ 0) ∧ (found = FALSE) then
        min_HB ← HB(i)
        found ← TRUE
    end if
    actual_BW ←| HB(i) − previous_HB |
    if actual_BW > max_BW then
        max_BW ← actual_BW
    end if
    previous_HB ← HB(i)
end for
found ← FALSE
for ((i = 255 → 0) ∧ (find = FALSE)) do
    if HB(i) ≠ 0 then
        max_HB ← HB(i)
        found ← TRUE
    end if
end for
real_HW ← max_HB − min_HB
if (real_HW < HW) ∧ (max_BW > BW) then
    IET ← HS
else
    IET ← HE
end if
```

output the image enhanced by the algorithm selected by the *Histogram Analyzer*.

In the following subsections all the implementation details of the SAFE modules are deeply analysed.

4.3.3.3.1 Histogram Analyzer

The *Histogram Analyzer* receives in input *HBs*, *HA* and *HD* signals from the *Histogram Calculator* IP-core. In addition, it receives the two SAFE input thresholds: *BW* and *HW*. This module extracts, from the image histogram associated with the previous frame, all the information required to calculate and select the best IET to be applied to the current frame. The internal architecture of the *Histogram Analyzer* module is shown in Figure 4.26.
It is composed of three main components: the *Histogram Buffer*, the *BW Calculator* and the *HW Calculator*.

The *Histogram Buffer* acts as namesake components in the *Histogram Stretcher* and *Histogram Enhancer*. However, differently from the other two IP-cores contained in this family, in SAFE the

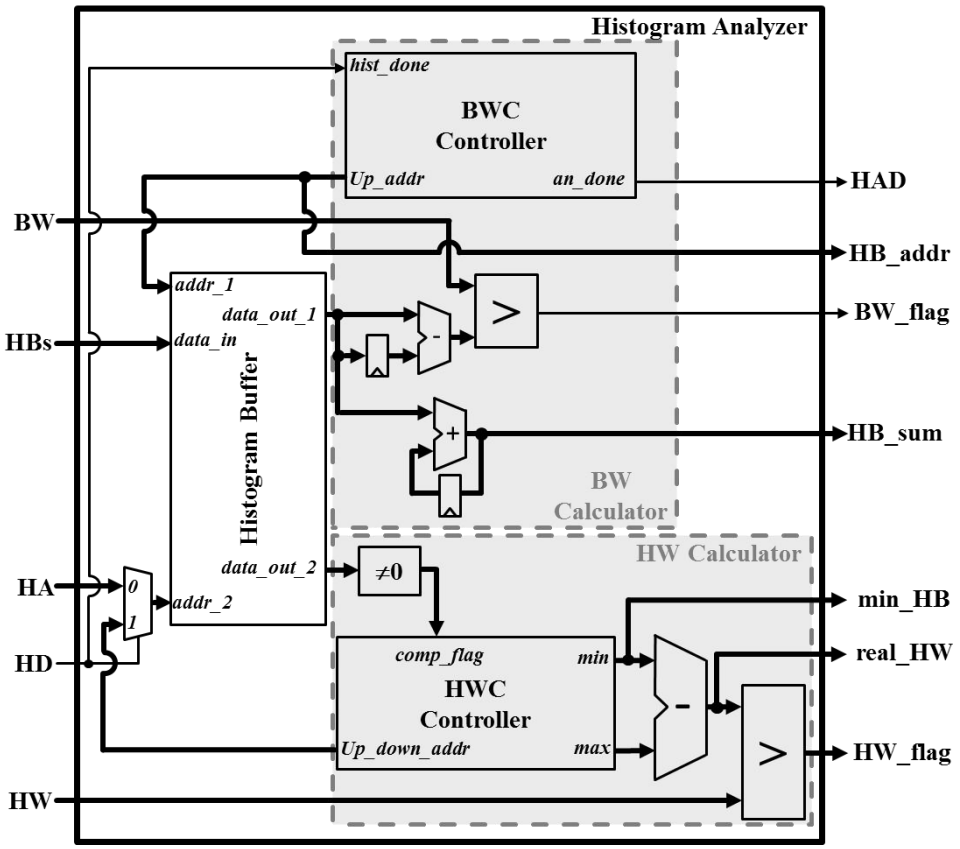


Figure 4.26: Histogram Analyzer internal architecture

Histogram Buffer is implemented as a dual-port BRAM. This different architectural choice, as will be described in the sequel, allows to parallelize the histogram analysis tasks.

When the entire histogram is stored, the *HD* signal is asserted and the histogram analysis can start. During this phase, *HW Calculator* and *BW Calculator* work in parallel, exploiting the dual-port nature of the *Histogram Buffer*.

The *HW Calculator* finds the maximum and minimum intensity value in the histogram, exploiting the same approach used in the *Histogram Stretcher* (Section 4.3.3.1). Minimum and maximum values are then subtracted to calculate the histogram width (i.e., *real_HW*). This quantity is then compared with *HW* threshold and, if it is greater, the *HW_flag* signal is asserted. The comparing task is performed exploiting a comparator with a parallelism equal to the bpp resolution of the input image, and the entire process is managed by the *HWC Controller*.

The *BW Calculator* scans the *Histogram Buffer* in order to find the maximum difference between two adjacent HBs. This quantity is compared with the *BW* threshold and, if it is greater, the *BW_flag* signal is asserted.

In parallel to this task, *BW Calculator* computes the *HB_sum* exploiting the same process used by the *Histogram Enhancer*(Section 4.3.3.2).
Finally, when the histogram scanning is completed, the *BW Controller* asserts the *HAD* signal.

The entire histogram analysis process requires just a number of clock cycles equal to the maximum intensity value in the image (2^{bpp}), since the histogram is composed of this number of HBs.

4.3.3.3.2 Equalizer / Stretcher

The *Equalizer / Stretcher* module receives in input, from the *Histogram Analyzer*, all the signals required to perform the enhancement of the input image, and it provides the enhanced pixels. The internal architecture is shown in Figure 4.27.

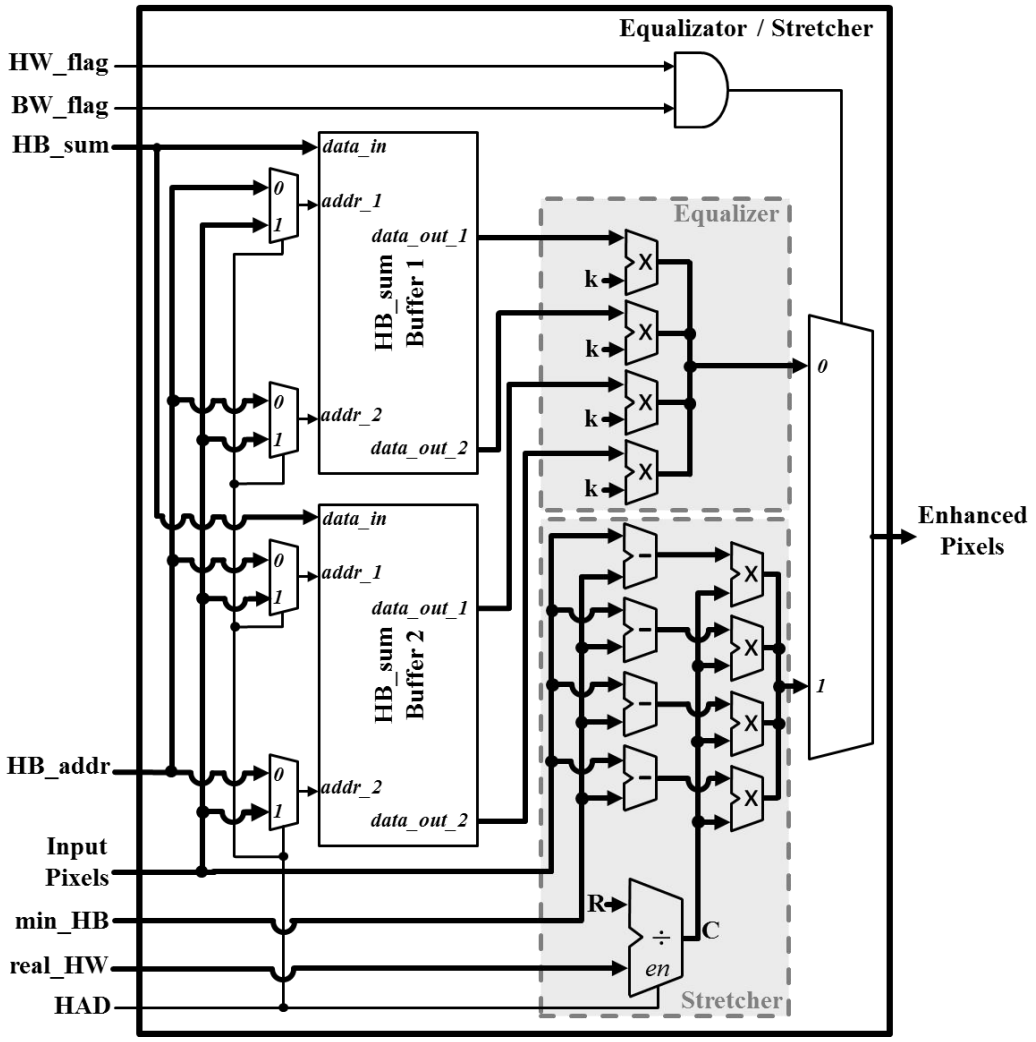


Figure 4.27: Histogram Equalizer and Stretcher internal architecture

This module is composed of two *HB_sum Buffers*, the *Equalizer* and the *Stretcher* modules.
These components operate in the same way as the namesake ones contained in the *Histogram Stretcher* and *Histogram Equalizer* IP-cores in order to compute the Equalized and Stretched pixel values. However, just the pixels enhanced with the best enhancement technique are provided in output.
This selection is done using a multiplexer in which the select signal is driven by the logic-and operator between *HW_flag* and *BW_flag*. In this way, the stretched pixels are output if *HW_flag* and *BW_flag* are asserted (i.e., the image histogram is narrow and very peaked), otherwise equalized pixels are provided in output.

The *Equalizer / Stretcher*, for enhancing the input image, requires the number of clock cycles needed to receive 1,024 x 1,024 pixels, plus 23 additional clock cycles to perform the C factor computation, because the divider is not a combinatorial component. Thus, the overall number of clock cycles required to enhance an image is equal to:

$$N_{clock} = \frac{N_{row} * N_{columns}}{4} + 23 \tag{4.16}$$

4.3.3.3.3 SAFE performances

In this section the performances of SAFE in terms of speed, area occupation and accuracy on a space qualified *Xilinx Virtex 5-QV XQR5VFX130* (Section 3.3) are evaluated. All the data reported herein concern a SAFE configuration for enhancing an composed of 1,024 x 1,024 pixels and with 8 bpp resolution.
Table 4.6 lists the area occupation on the target device, highlighting the resources usage of each SAFE module.

Table 4.6: Resource Usage for *Xilinx Virtex 5-QV XQR5VFX130*

Module	**FPGA Area Occupation**			**Max Freq.**
	Slices	**DSPs**	**BRAMs**	**[MHz]**
HA	219 (0.26%)	- (-)	2 (0.67%)	75.28
E/S	434 (0.53%)	20 (6.25%)	4 (1.34%)	72.69
Total	653 (0.79%)	20 (6.25%)	6 (2.01%)	

The maximum input frame rate that can be sustained by SAFE is related to the number of clock cycles needed to analyse and enhance the input images. The time slot required between the acquisition of two consecutive frames is equal to the time required to analyse the image histogram, and to calculate the scale factor C for the image stretching. Since the maximum operating frequency is equal to 72.69 MHz, the time required to accomplish these tasks is:

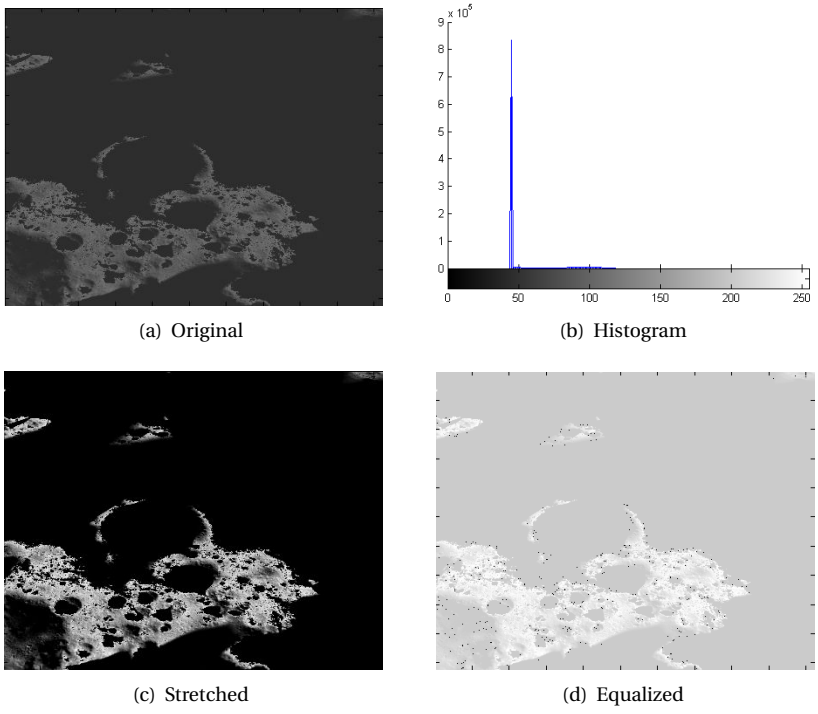
$$T_{exec} = \frac{N_{CLK\_HA} + N_{Equalizer/Stretcher)}}{f_{MAX}} =$$

$$= \frac{256 + 262,167}{72.69\,MHz} = 3.61ms \tag{4.17}$$

This execution time led to a maximum frame rate of 277 fps.

This result clearly shows that, in an images processing chain, SAFE far exceeds the typical frame rate of 25 fps of the other building blocks, and of the camera, as well.

Post place and route simulations have been done using Modelsim SE 10.1c [83], and results have been validated comparing them with the values obtained by a MATLAB model of the described architecture. The validation involved a set of 50 images of planetary environments provided by *Thales Alenia Space* company. Figure 4.28 and Figure 4.29 show examples of frames enhanced by the two available algorithms. In Figure 4.28 the best result is achieved by applying the HS, while in Figure 4.29 the HE should be preferred. Starting from the original frame shown in Figure



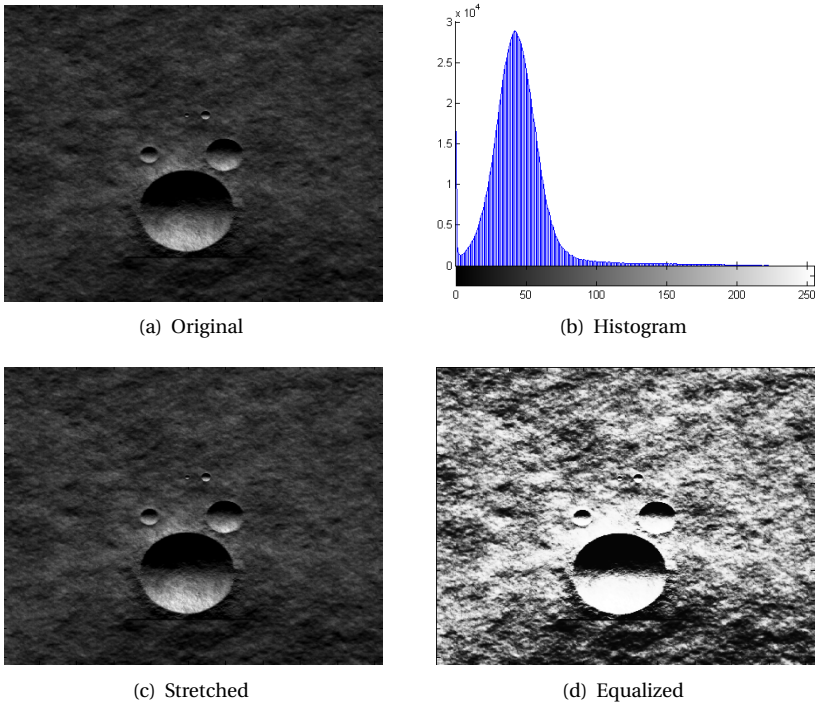
(a) Original (b) Histogram

(c) Stretched (d) Equalized

Figure 4.28: Test of frame enhancement (best results by HS)

4.28(a), the relative histogram is computed. As shown in Figure 4.28(b), in this case the histogram is narrow and extremely peaked. Thus, a HS approach should be preferred, since it outputs an image with high contrast (Figure 4.28(c)), with respect to the equalized one (Figure 4.28(d)).

By analysing the original frame shown in Figure 4.29(a) and the associated histogram (Figure 4.29(b)), one can notice that the histogram is smoothed and relatively wide. Thus, the HE will give better results in term of image contrast, as can be inferred by visually comparing Figure



(a) Original (b) Histogram

(c) Stretched (d) Equalized

Figure 4.29: Test of frame enhancement (best results by HE)

4.29(d) and Figure 4.29(c).
After tuning *HW* and *BW* thresholds, the validation campaign has demonstrated that SAFE is always able to select the best IET to be applied on the input frames. Moreover, comparisons have been done between frames enhanced with SAFE (i.e., exploiting the information associated with the previous frame), and frames enhanced with a standard approach (i.e., exploiting the information of the associated histogram). This comparison showed that SAFE introduces just a maximum error of 0,39% on the intensity of the pixels, w.r.t. the standard approach.

4.3.4 Feature Matchers

As described in Section 2.5.1.2, a fundamental step for relative VBN systems is the features matching task. For this reason, *Features Matchers* family groups a set of high-performance IP-cores implementing the most important area-based features matchers (Section 2.5.1.2.1). In particular the features matchers contained in this family are: the *Sum of Absolute Difference* (SAD), *Sum of Square Difference* (SSD), *Cross-Correlation* (CC), and *Normalized Cross-Correlation* (NCC).

The major common benefit of the proposed implementations is the complete independence of the hardware resources usage from the image window used to find correspondent features between the two images to be matched. This peculiarity ensures to increase the research window,

so enhance the matching precision, without increasing the required resources.
To achieve this goal, the IP-cores of this family shares a common basic architecture (Figure 4.30).

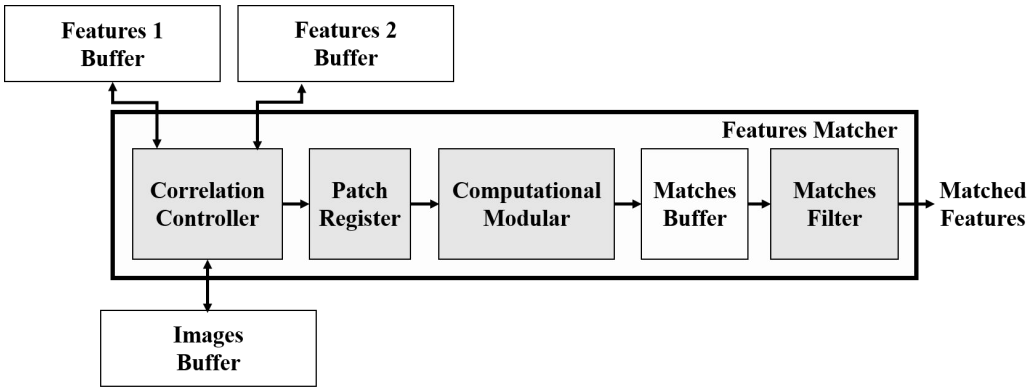


Figure 4.30: Feature matcher internal architecture

The *Features Matcher* is able to communicate with three external buffers that can be freely implemented inside the FPGA as BRAMs or as storage areas in an external memory. These buffers aim at storing: the features associated with the images to be matched (*Features 1 Buffer* and *Features 2 Buffer*), and the two images to be matched (*Images Buffer*).

The *Correlation Controller* scans the *Frame 1 Features* buffer and the *Frame 2 Features* and for each couple compute the matching operation (SAD, SSD, CC, or NCC). The matching operation is computed using the intensity of all pixels contained in the two pixels windows surrounding the two features (i.e., matching windows). These values are loaded from the *Images Buffer*.
The matching window related to the first feature is loaded from the *Features 1 Buffer* into the *Patch Register*, that is composed of a number of registers equal to the number of pixels composing the matching window. Then, while the window associated with the feature of the second image is loaded from *Features 2 Buffer*, the matching operation is computed "on-the-fly". Each time a new pixel is received from the *Images Buffer*, it is computed the matching operation between this pixel and the corresponding one of the first image, that is already stored in the *Patch Register*. This operation is performed by the *Computation* module that implements the desired matching operation. This approach makes the area occupation of this module independent from the correlation window dimension, making the designer free to select the more appropriate correlation window without any area occupation penalty.

The matching operation results are thresholded, in order to eliminate fake-matchings. If the obtained result identify a valid match (i.e., the procedure to identify a valid match changes depending on the implemented matching operation (Section 2.5.1.2)), the coordinates of the correlated features are stored inside the internal *Matched Buffer*, implemented exploiting BRAMs. The

size of this buffer in terms of BRAMs depends on the maximum number of expected matched features pairs, so the designer can freely configure this size depending on the target application.

Finally, since a feature of the first image can be correlated to several features of the second image, only the match that has the highest probability to be valid is provided in output, ensuring unique matched pairs and higher quality of matches. This final filtering operation is computed by the *Matches Filter* module.

The described process is executed by every IP-core composing this family. The unique differences among them are: the way in which the matching operation is compute, and the way to consider valid a matched depending on the matching operation result. The following sections provides a description of the *Computational Module* implementation for each IP-core.

4.3.4.1 SAD

As described in Section 2.5.1.2, *Sum of Absolute Difference* (SAD) based matcher is the area-based features matching approach with the lowest complexity. For this reason, the associated hardware implementation is the simplest one (Figure 4.31).

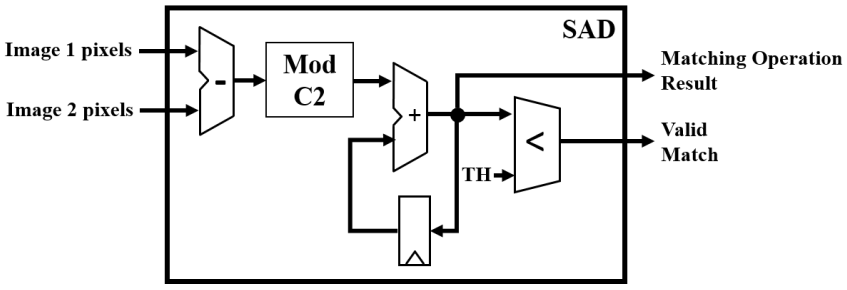


Figure 4.31: Internal architecture of the *Computational Module* implementing the *Sum of Absolute Difference* based matching

According to Equation 2.10, this module receives in input the pixels, associated with the two images to be matched, contained in the reference window. Each couple of received pixels is subtracted. To take into account negative values from the subtraction, the pixels are internally represented with the complement 2 format. Each difference is then made positive by means of a complement 2 absolute value computation (*C2 mod* component in Figure 4.31), and the accumulator sum together these positive contributions. When all the pixels contained in the reference windows have been received, the accumulator contains the SAD based matching result. This value is finally compared with an internal threshold, and, if it is lower, it is provided in output as a valid match (the *valid match* signal is asserted).
Thanks to the proposed implementation, the computation can be done in a pipelined way, without introducing additional latency in the processing chain.

Table 4.7 reports the area occupation and the speed of the proposed SAD based matcher IP-core implemented on a space qualified *Xilinx Virtex 4-QV XQR4VLX200* (Section 3.3). The reported data refers to the IP-core configured to match features between two 1,024 x 1,024 pixels images with a bpp resolution of 8 bit, based on a 11x11 reference window.

Table 4.7: *SAD based Features Matcher* performances on a *Xilinx Virtex 4-QV XQR4VLX200*

FPGA Area Occupation				**Max Freq.**
Slices	**LUTs**	BRAMs	**DSPs**	**[MHz]**
700 (0.79%)	1,185 (0.67%)	5 (1.49%)	- (-)	94.76

The reported data demonstrates how the simplicity of the SAD based matching operation is reflected in an extremely low area occupation.

4.3.4.2 SSD

The proposed *Sum of Square Difference* (SSD) based matcher IP-core implements Equation 2.11 in an efficient way. As shown in Figure 4.32, this module processes the received pixels in a similar way to SAD based matcher. However, in order to make the difference values positive, it computes the square value instead of the absolute one. This modification is accomplished by replacing the *mod C2* component with a multiplier receiving on both inputs the computed difference values.

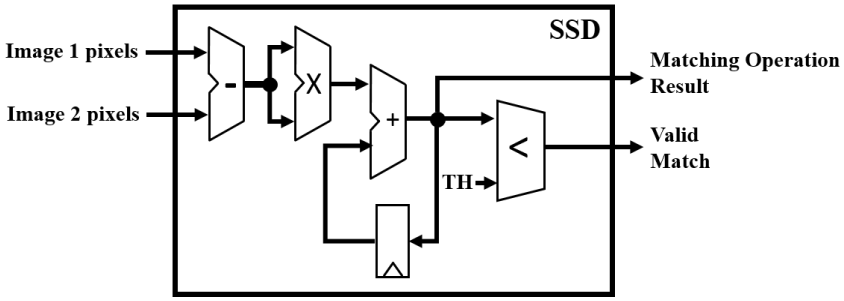


Figure 4.32: Internal architecture of the *Computational Module* implementing the *Sum of Square Difference* based matching

Table 4.8 reports the area occupation and the speed of the SSD based matched IP-core implemented on a space qualified *Xilinx Virtex 4-QV XQR4VLX200* (Section 3.3). The data refers to the IP-core configured to perform the matching task based between two 1,024 x 1,024 pixels images with a bpp resolution of 8 bit, based on a 11x11 reference window.
As can be noted from the reported data, the usage of the multiplier to make the computed difference positive leads to a higher area occupation than the SAD based matcher. The speed is the

Table 4.8: *SSD based Features Matcher* performances on a *Xilinx Virtex 4-QV XQR4VLX200*

FPGA Area Occupation				**Max Freq.**
Slices	**LUTs**	BRAMs	**DSPs**	**[MHz]**
743 (0.84%)	1,254 (0.70%)	5 (1.49%)	- (-)	94.76

same in these two components, since the critical path is contained in the common architecture part.

4.3.4.3 CC

As described in Section 2.5.1.2.2, the *Cross-Correlation* (CC) based matching approach uses a different approach than the ones based on pixel differences (SAD and SSD based matching operations). For this reason the internal architecture of this component (Figure 4.33) present two main differences w.r.t. the two previously presented ones.

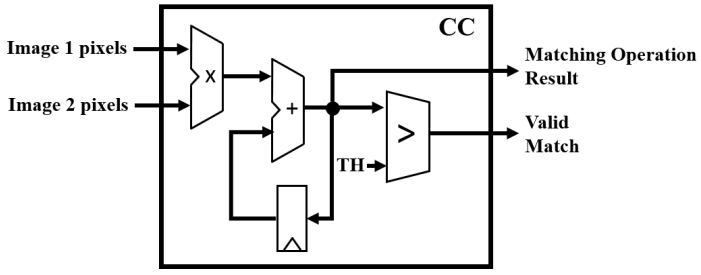


Figure 4.33: Internal architecture of the *Computational Module* implementing the *Cross-Correlation* based matching

The former concerns the way to compute the matching results. In this component, according to Equation 2.12, instead of subtracting the received pixels values, the corresponding inputs are multiplied together. Then, the multiplication are sum by means of an accumulator.

Instead the latter is related to the way to discriminate a valid from an invalid match. In fact, as already mentioned in Section 2.5.1.2.2, a correlation based matching provide a high result when the probability that two features are correlated is really high. For this reason, in this component, to properly define valid matches through the *valid match signal*, the matching result contained in the accumulator is compared with an internal threshold by means of a higher-than comparator.

Table 4.9 reports the area occupation and the speed of the CC based matched IP-core implemented on a space qualified *Xilinx Virtex 4-QV XQR4VLX200* (Section 3.3). The data refers to the IP-core configured to perform the matching task based between two 1,024 x 1,024 pixels images with a bpp resolution of 8 bit, based on a 11x11 reference windows.
As can be noted, even if this architecture requires a component less than the SSD based matcher,

Table 4.9: *CC based Features Matcher* performances on a *Xilinx Virtex 4-QV XQR4VLX200*

FPGA Area Occupation				**Max Freq.**
Slices	**LUTs**	BRAMs	**DSPs**	**[MHz]**
734 (0.82%)	1,242 (0.70%)	5 (1.48%)	- (-)	94.76

the area occupation is similar. This is mainly due to negligible area occupation resources usage associated with the *mod C2* component. Concerning the speed, also in this case the critical path is in the common part architecture, so the maximum working frequency is the same of the difference based matchers.

4.3.4.4 NCC

As described in Section 2.5.1.2, *Normalized Cross Correlation* (NCC) based matcher is the area-based features matching approach with the highest complexity. This is reflected in an extremely more complex hardware architecture (Figure 4.34)than the previously presented matchers.

This IP-core is composed of two modules working in parallel: *Num* and *Den*. These two modules implement respectively the numerator and denominator of Equation 2.13.

Num performs the same operations done by the *CC* based matcher to obtain the matching result.

Instead, *Den*, first, computes the two terms (i.e., $T_1 = \sum_{i=0}^{M}\sum_{j=0}^{N}(I_1(x_{f1}+i, y_{f1}+j))^2$ and $T_2 = \sum_{i=0}^{M}\sum_{j=0}^{N}(I_2(x_{f2}+i, y_{f2}+j))^2$) composing the denominator of Equation 2.13 by means of two multipliers and two accumulators. Then, it multiplies together T_1 and T_2 values, and, finally, the denominator value is obtained by compute the square root of the multiplication result (*sqrt* module in Figure 4.34).

Table 4.10 reports the area occupation and the speed of the NCC based matched IP-core implemented on a space qualified *Xilinx Virtex 4-QV XQR4VLX200* (Section 3.3). The data refers to the IP-core configured to perform the matching task based between two 1,024 x 1,024 pixels images with a bpp resolution of 8 bit, based on a 11x11 reference windows.

Table 4.10: *NCC based Features Matcher* performances on a *Xilinx Virtex 4-QV XQR4VLX200*

FPGA Area Occupation				**Max Freq.**
Slices	**LUTs**	BRAMs	**DSPs**	**[MHz]**
1,817 (2.04%)	2,729 (1.53%)	6 (1.79%)	18 (18.75%)	83.57

The reported data demonstrates the really high complexity of this matching method, that is directly reflected in extremely higher area occupation. Moreover, differently from the previously

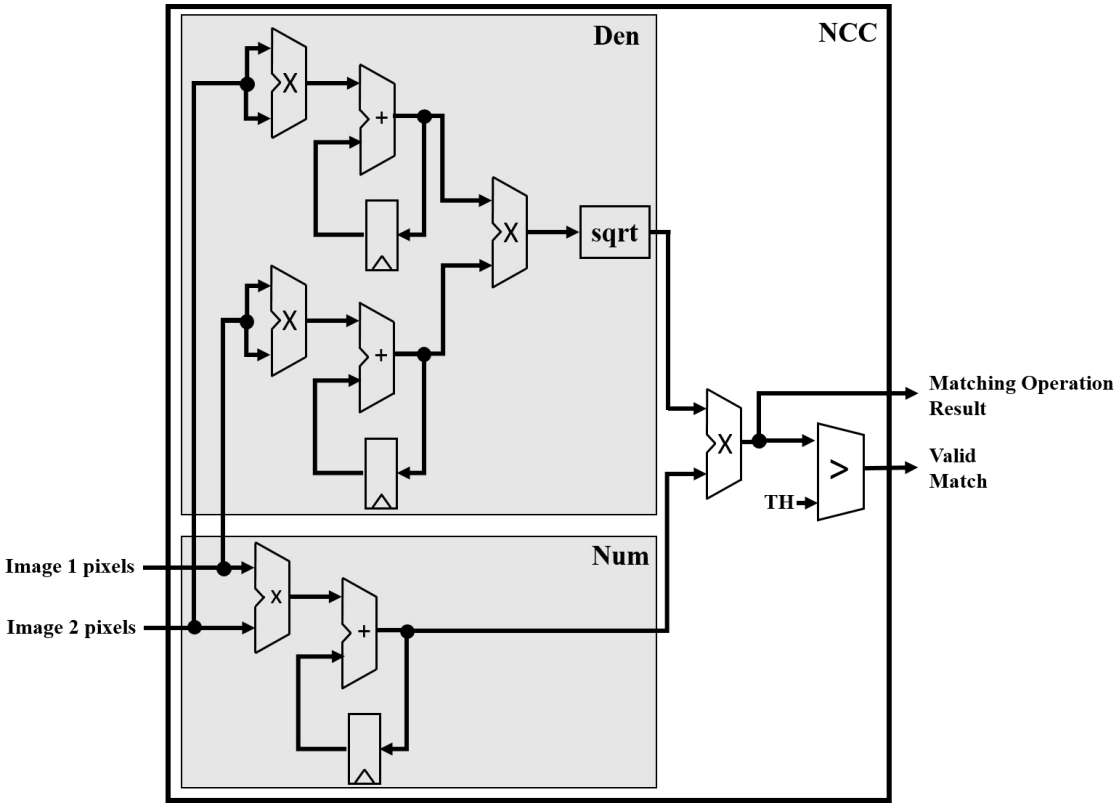


Figure 4.34: Internal architecture of the *Computational Module* implementing the *Normalized Cross-Correlation* based matching

presented architecture, in this component the computational module increase the critical path duration, leading to a lower working frequency.

CHAPTER 5

COMPLEX IMAGE PROCESSING SYSTEMS IN SPACE

This chapter presents the proposed complex image processing systems for space applications. These systems, result of strong collaboration between the academic world and the industries operating in the space field, accomplish the current image processing needs of space agencies. They are able to provide the required performances, in terms of hardware resources usage, speed and accuracy, exploiting the FPGA-based hardware accelerators presented in Chapter 4. In addition to these IP-cores, these systems embeds innovative ad-hoc hardware components and software routines able to provide high performance and self-adaptable image processing functionalities. These elements have not been inserted inside the library since they aim at solving specific task that cannot be considered of general purpose as the functionality provided by the library components.

In this chapter three main systems have been proposed to cover the needs of space agencies in the video-based navigation, the active space debris removal, and the image compression. In particular, Section 5.1 presents two hardware systems resulted from a strong collaboration with *Thales Alenia Space Italy*, that allow to strongly accelerate the feature extraction and matching task (i.e., one of the most computational intensive task of the relative video-based navigation), while requiring limited hardware resources usage and providing high adaptability to environmental conditions.
Section 5.2 proposes an innovative systems merging hardware components and software routine to compute in real-time the debris characterization task (i.e., the first phase required by the active space debris removal). This system has been realized in the framework of a strong collaboration among *Politecnico di Torino*, *Istituto Italiano di Tecnologia*, and *Eurix s.p.a.*.
Eventually, Section 5.3 presents the main outcomes of my visiting period at the *European Space and Technology Centre* (ESTEC), that is the main research centre of the *European Space Agency* (ESA). The developed system aims at efficiently mapping the JPEG-LS image compression algorithm (Section 2.7.1) on an FPGA device to strongly enhance the timing performances of the image compression performed during a space exploration mission.

In the following sections, the internal architecture of each proposed system is deeply described. Moreover, to highlight the benefits in terms of performances and self-adaptability to the environmental conditions, a comparison of the proposed systems with the current state-of-the-art implementations is provided.

5.1 FPGA-based hardware accelerators for Video-Based Navigation in space applications

As described in section 2.5.1.1, a VBN system employed during an EDL phase must provide highly accurate spacecraft speed and position. To ensure this precision, ME algorithms require very accurate matching points distributed across the entire frame [127]. While ME algorithms are not computationally intensive, FEM algorithms require high computation capability to guarantee high frame rates and therefore high accuracy. Hence, very efficient hardware accelerators for this task are mandatory.

The most efficient feature extractor for a VBN system, according to section 2.5.1.1, are: Harris corner detector [86], SURF [17] and SIFT [128]). From the algorithmic point of view, SURF and SIFT are probably the most robust solutions since they are scale- and rotation-invariant. This means that features can be matched between two consecutive frames even if they have differences in terms of scale and/or rotation. However, due to their complexity (Section 2.5.1.1.2 and 2.5.1.1.3), hardware implementations are very resource hungry. As an example, [16] and [25] propose two FPGA-based implementations of the SURF algorithm. The architecture proposed in [16] consumes almost 100% of the LUTs available on a medium sized *Xilinx Virtex 6* FPGA, without guaranteeing real-time performances. Similarly, the architecture proposed in [25] consumes about 90% of the internal memory of a *Xilinx Virtex 5* FPGA. It saves logic resources, but it is able to real-time process images with a resolution limited to 640 x 480 pixels. Another example is presented in [233], where an FPGA-based implementation of the SIFT algorithm is presented. It is able to real-time process 640 x 480 pixel images, consuming about 30,000 LUTs and 97 internal DSPs in a *Xilinx Virtex 5* FPGA. Comparing these data with the available hardware resources in space-grade *Xilinx Virtex* FPGAs (Section 3.3), it can be immediately noted that no space is available to apply SEU mitigation techniques (Section 3.4) that are mandatory to use a *Xilinx Virtex* FPGAs in space environment.

Instead, Harris corner detector provides a better trade-off between precision and complexity [211]. Under the assumption of small differences between consecutive frames (i.e., high frame rates or small camera displacements), its accuracy is comparable to SURF and SIFT, with a significant lower complexity. This decrease in complexity can be immediately appreciated if the reader compares the operations required by Harris corner detector (Section 2.5.1.1.1) with the ones required by SURF (Section 2.5.1.1.3) and SIFT (Section 2.5.1.1.2). Since high frame-rates are mandatory during the EDL phase to allow real-time correction of the descending trajectory, Har-

ris is a very good candidate to implement a high-speed and low-area FEM accelerator block for space-applications [58].

This chapter describes the results of my research work on VBN systems for space applications. In particular, it presents FEMIP and *Self-Adaptive* FEMIP (SA-FEMIP). The former is a high performance FPGA-based FEM IP-core based on the Harris algorithm and optimized for being used in space EDL systems. The latter is an improved version of FEMIP that enables the self-adaptation of the internal algorithm parameters. Self-adaptation is a key aspect to better optimize the FEM algorithm to the environmental conditions, thus increasing the robustness with respect to noise and variations of external conditions that are typical of the space environment.

An FPGA implementation has been preferred to follow the current trend that is replacing ASICs with more flexible FPGA blocks in several mission critical applications [85].
Only few publications proposed FPGA-based implementations of the Harris algorithm. Benedetti et al. [19] presented a very high speed hardware architecture (i.e., 30 fps output frame rate). However, their solution requires the parallel use of four FPGAs and is therefore not suitable for space-applications.
Cabani et al. presented in [26] an interesting scale-invariant implementation of Harris. However, area occupation is not well-optimized.
To the best of our knowledge, the state-of-the-art solution has been developed at the University of Dundee in the framework of the *ESA NPAL Project* [58]. It provides good performances in terms of resources utilization and output frame rate (20 fps). Nonetheless, it requires an external co-processor to perform the matching phase. The present IP-cores overcome these limitations, proposing high performance FEM architectures able to increase the sustainable input frame rate and the adaptability to the unpredictable environmental conditions of space environment, with very limited hardware resources and without resorting to external co-processors.
The rest of this chapter deeply describe the internal architecture of FEMIP [53] and SA-FEMIP [52], highlighting the benefits of the proposed solutions in terms of accuracy, timing performance and hardware resources usage.

5.1.1 FEMIP

The overall FEMIP architecture is reported in Figure 5.1. It gets a 32-bit input stream representing 1,024 x 1,024 grey scale images with 10 bpp resolution, as provided by almost all space-qualified CMOS cameras [70]. It provides a set of features that match between two consecutive images. FEMIP internal structure includes three functional blocks: *Gaussian Filter*, *Harris Features Extractor*, *Features Matcher*.

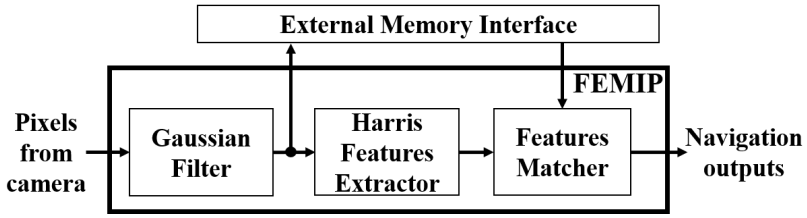


Figure 5.1: FEMIP complete chain

5.1.1.1 Gaussian Filter

The *Gaussian Filter* performs Gaussian smoothing [82] of the input image. It reduces the level of noise in the image, improving the accuracy of the feature extraction algorithm. In our architecture, Gaussian filtering is performed via a 2D-convolution of the input image with a 7x7 pixels Gaussian kernel mask [82]. A 7x7 kernel is enough to forcefully reduce the noise that strongly affects images taken in space environments.

The 2D-convolution has been implemented in hardware exploiting the *2D-convolver* presented in Section 4.3.1.1. Since this component has an input interface able to receive one pixel at the time, an input controller has been added. This module receives the image flow through the 32-bit FEMIP input interface and unpacks it in order to reconstruct the original 10-bit pixel flow. Obviously, the *2D-convolver* has been configured in terms of internal buffers size and number of algebrical components (i.e., multipliers and adders) to implement the 2D-convolution with the required configuration. Thus, since a 7x7 pixel Gaussian kernel is needed, RB has been implemented with 7 FPGA Block-RAMs (BRAMs), while the *Sliding Window Buffer* (SWB) is made up of 49 10-bit registers. Eventually, the MUL/ADD tree implementing the *Computation Stage* contains 49 multipliers and 6 adder stages, for a total of 48 adders. Adders and multipliers must support fixed-point representation, because the used kernel values are fractional numbers. The kernel content has been represented in the 0.15 format, assuring a minimal error introduced by the fixed-point approximation (Section 5.1.1.4).

The output filtered pixels obtained from the *2D-convolver*, represented in 10.15 bit format, are sent both to the *Harris Features Extractor* and to an external memory via a second 32-bit output interface. Storing filtered pixels in an external memory is mandatory since this information is needed during the following features matching phase.

5.1.1.2 Harris Features Extractor

The *Harris Features Extractor* implements the Harris corner detection algorithm [86], and applies it on the filtered pixels received from the *Gaussian Filter* block. It outputs a set of extracted features represented as: *feature coordinates* and the related *R-factors* (i.e., *R(x,y)*). For detail information about he Harris corner detector algorithm, the reader may refer to Section 5.1.1.2.

Figure 5.2 shows the internal architecture of the *Harris Features Extractor*.

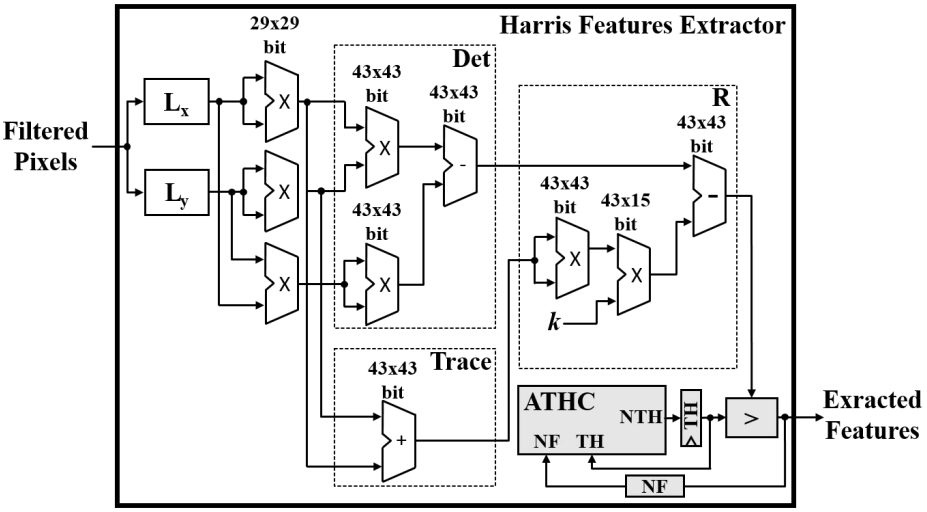


Figure 5.2: *Harris Features Extractor* internal architecture

The first two modules, L_X and L_Y, compute the spatial image derivatives on the filtered image in the horizontal and vertical direction, respectively. This operation is a 2D-convolution of the filtered image with the 3x3 Prewitt kernel [82]. Equation 5.1 reports the G_x and G_y 3x3 Prewitt kernels, useful to compute the image derivatives in the horizontal and vertical direction, respectively.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} \tag{5.1}$$

Prewitt kernel factors are represented according to the 0.15 format, adding one extra bit in order to use the 2's complement representation, required to take into account negative factors. In order to achieve high performance, these two modules implement the same pipelined architecture exploited in the *Gaussian Filter*, with just two differences. First, the RB is adapted to the 3x3 dimension of Prewitt kernel. Six BRAMs, two for each filtered image row, are required to account for the increased bit-per-pixel representation used after filtering. The RB is in common between L_X and L_Y modules. Second, the MUL/ADD tree only contains six 25x16 bit fixed-point multipliers (just 6 values are different from zero in Prewitt kernels) and 3 adder stages, for a total amount of 6 fixed-point adders.

The next 29x29 bit multipliers compute the coefficients of the second-moment matrix, which are required to calculate the Harris *R-factor* associated with each filtered image pixel (Equation 2.1).

The algorithm characterization (Section 4.1) performed on the *Harris Features Extractor*, high-

lighted that the maximum values obtained during this computation can be represented resorting to a 28.15 format. Thus, the output of 29x29 multipliers and of all following blocks of the *Harris Features Extractor* (Figure 5.2), that implement Equation 2.1, can be truncated to this format, resulting in a strong area reduction.

Finally, the 43-bit comparator at the output of this module applies a threshold on each computed *R-factor*. It outputs just those features whose *R-factor* is greater than a given threshold together with the coordinates of the associated pixel. This guarantees that only the features that potentially represent a real corner are propagated to the next module.

The value of the threshold strongly depends on the image environment type (e.g., Mars or Moon) and condition (e.g., brightness, noise or contrast). To increase adaptation, a self-adaptive threshold is computed by the *Adaptive Threshold Computation Module* (ATHC) shown in Figure 5.2.

ATHC performs the thresholds adaptation routine reported in Algorithm 4 after the analysis of each image, calculating the threshold to apply to the following image. It receives the number of features extracted from the current image (NF) and the current threshold (TH), initialized at 0 at startup (i.e., all features are accepted). The number of selected features (NF) is compared with the number of expected features (TF), in our specific test application set to 3,000. If the two numbers are equal with a tolerance defined by δ the threshold is already optimized. If not, a new threshold is computed adding *Step* to the current value of TH.

Algorithm 4 Threshold adaptation routine

```
Require: NF                                   ▷ Number of computed features
Require: TH                                   ▷ Current threshold
  Const TF=3000                               ▷ Number of features to select
  Const δ=50                                  ▷ Tolerance
  Disp=NF-TF
  Step = Disp * (0.5/TF)*TH
  if Disp< −δ OR Disp> +δ then
     NTH=TH+Step
  else
     NTH=TH
  end if
  return NTH
```

Algorithm 4 computes the threshold for the next image based on information on the current image. This is acceptable since, thanks to the high frame rate achievable by our architecture, consecutive frames show marginal differences. Obviously, at startup, ATHC needs some cycles to output a valid threshold. During this phase, the *Features Matcher* is unable to produce useful results. However, an experimental campaign on a set of planetary images provided by *ThalesAlenia Space Italy S.p.a.* (*Verification Dataset* in Section 5.1.1.4), highlighted that the maximum number of frames required to reach a stable threshold is relatively small (13 frames). After this transitory phase, TH becomes stable and the *Features Matcher* can start processing the extracted features,

while ATHC continuously adjusts the threshold depending on the sampled images.

5.1.1.3 Features Matcher

The *Features Matcher* (Figure 5.3) receives the features extracted by the *Harris Feature Extractor* and finds the set of features that match in two consecutive images.

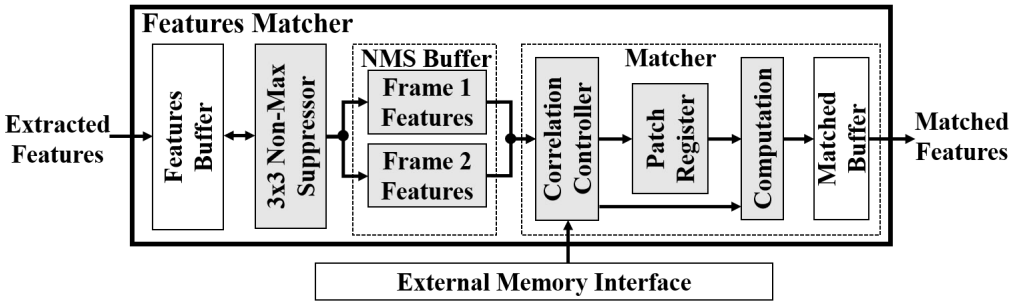


Figure 5.3: *Features Matcher* internal architecture

This module adopts two different optimization strategies. The former concerns the matching task, that is performed exploiting un-normalized Cross-Correlation. The un-normalized Cross Correlation is a matching approach less robust to variations of environmental conditions than the Normalized Cross-Correlation (NCC) one (Section 2.5.1.2.1). Although, in this context the high frame rate leads to negligible differences in the conditions (e.g., brightness or contrast) of two consecutive images. Thus, the usage of un-normalized Cross Correlation does not introduce any error in the matching task. In addition, if compared to a NCC approach [241], it leads to a very simple hardware implementation, providing a significant gain in FPGA resources utilization and throughput.
The latter concerns the selection of potentially correlated features. Analysing the speed of a space-module during the descending phase, and considering the high input frame rate used to sample images, we identified that a feature can perform a maximum movement of 17 pixels between two consecutive images [203]. Thus, two features can be considered as potentially correlated if they are both in a 35x35 pixel neighborhood between the two considered images. Cross-Correlation is therefore computed on these features, only, reducing the computational load and speeding up the matching task.

The *Features Matcher* receives feature coordinates and associated *R-factor* from the *Harris Feature Extractor*, and stores them in the *Features Buffer* (FB), implemented as a group of BRAMs. In our specific implementation, this buffer can store up to 3,500 features, using 14 internal BRAMs. Whenever an entire image is processed and all features are stored in FB, the *3x3 Non-Max Suppressor* performs a preliminary filtering operation. For each feature, it scans a 3x3 pixels neigh-

bourhood looking for close features. If they are found, the feature with the highest *R-factor* in this region is marked as valid, while the others are marked as non-valid. To speed up this operation, that would require a complete search into FB, we observed that, in our experimental campaign, no more than 10 features per image row have been identified. Given this observation, considering that features are obtained analysing the image row by row and then saved into FB, a neighbour feature will be for sure stored in a (+20, -20) region of FB, centred on the considered feature. This allows us to reduce the neighbour search space and therefore to dramatically decrease the execution time, without increasing area occupation.
All valid features are stored in the the *NMS Buffer*, that can store up to 1,000 filtered features coordinates, using 4 BRAMs.

The *NMS Buffer* is composed of two sub-buffers (*Frame 1 Features* buffer and *Frame 2 Features* buffer). These two buffers are alternatively used to internally store features associated with two consecutive images, that must be analysed and matched. So that no external memory is required to store these information.

The *Correlation Controller* scans the *Frame 1 Features* buffer and the *Frame 2 Features* buffer looking for two correlated features. It compares the coordinates associated with a feature contained in one of the two buffers with all the coordinates in the other buffer. Whenever two potentially correlated features are found (i.e., their distance is no more than 17 pixels, as aforementioned), their un-normalized Cross-Correlation is computed using the intensity of all pixels contained in the two 11x11 pixels windows surrounding the two correlated features. These values (previously stored by the *Gaussian Filter*) are loaded from the external memory.
A 11x11 pixels Cross-Correlation window size has been chosen for the hardware implementation after a test campaign on planetary image sequences, that simulate the descending phase of a spacecraft. The internal implementation of this module follows the one presented in Section 4.3.4.3. In particular, for each pair of consecutive frames, the matched features have been saved in order to evaluate the number of true and fake matches. Figure 5.4 shows the maximum rate of fake matches, out of the number of total matches, that was observed between different images, varying the size of the Cross-Correlation window. Results have been evaluated by mean of an automatic script, able to detect fake matches between two consecutive images of the well-known descending test-cases.
As can be seen in Figure 5.4, a window size greater than 11x11 pixels does not provide any significant improvement on the quality of the matched couples. Furthermore, this implementation provides more than 9x precision improvement, compared to the current state-of-the-art [58], which is based on a 7x7 pixels window.
The 11x11 window related to the first feature is loaded into the *Patch Register*, that is composed of 121 25-bit registers. Then, while the window associated with the feature of the second image is loaded, the cross-correlation is computed "on-the-fly". Each time a new pixel is received from the external memory, it is multiplied from the corresponding pixel of the first image, that is al-

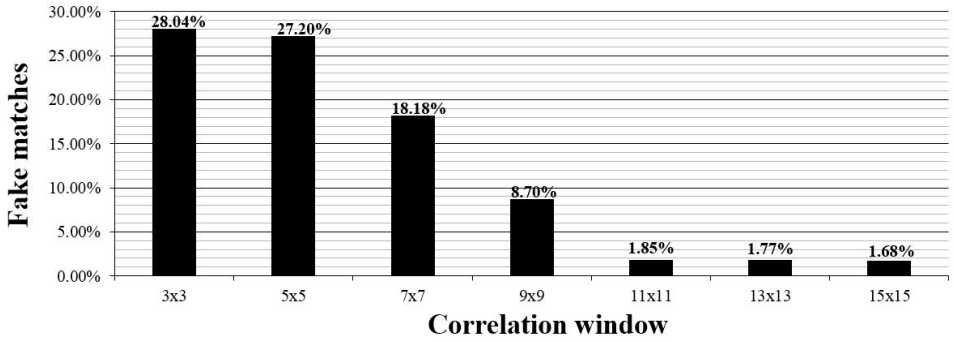


Figure 5.4: Fake matches on test images ranging different Cross-Correlation window size

ready stored in the *Patch Register*. This operation is performed by the *Computation* module that contains a 25-bit multiplier connected to an accumulator. This approach makes the area occupation of this module independent from the correlation window dimension, making the designer free to select the more appropriate correlation window without any area occupation penalty.

Finally, the Cross-Correlation results are thresholded, in order to eliminate fake-matchings. If the calculated Cross-Correlation value is higher than a given threshold, the coordinates of the correlated features are stored inside the internal *Matched Buffer*, implemented as a single BRAM. This buffer is able to store up to 512 matched features pairs. Moreover, since a feature of the first image can be correlated to several features of the second image, only the match that has the highest Cross-Correlation value (i.e., the highest probability to be correlated) is considered valid. This ensures unique matched pairs, and higher quality of matches.

5.1.1.4 Experimental results

To evaluate the hardware resources usage and the timing performances, the proposed architecture has been synthesized and implemented, resorting to *Xilinx ISE Design Suite 14.6* [229], on a space-qualified *Xilinx Virtex 4-QV VLX200* (Section 3.3) that, together with the *Xilinx Virtex 5-QV VFX130* (Section 3.3), represents the state-of-the-art architecture for space-qualified reprogrammable FPGAs (Section 3.3). The reason to select the *Xilinx Virtex 4* architecture instead of the newer *Xilinx Virtex 5* is twofold. First the SA-FEMIP architecture has been designed to be integrated and tested inside the *Thales Alenia Space Avionic Testbench* (ATB), i.e., a hardware infrastructure emulating the on-board computing platform of a spacecraft. The ATB is equipped with a *Gaisler Research GR-CPCI-XC4V* development board [78]. This board integrates a *Xilinx Virtex 4 XC4VLX200VLX200* [226], which provides the same internal logic architecture of the space-qualified version. Second, implementing FEMIP on a *Xilinx Virtex 4* FPGA allowed us to perform fair comparisons with other published architectures, thus highlighting the benefits of the introduced improvements.

Post place and route simulations have been done with Modelsim SE 10.1c [83].

Table 5.1 shows the performances of FEMIP in terms of area occupation, speed, and power consumption. The power consumption of each module reported in Table 5.1 does not take into account the contribution of the clock circuitry and the leakage. These contributions are included in the overall power consumption.

Table 5.1: Resources usage and power consumption of *FEMIP* implemented on a *Xilinx Virtex 4 XC4VLX200*

	Module	**FPGA Area Occupation**			**Max.Freq.**	**Power**
		Registers	**LUTs**	**BRAMs**	**[MHz]**	**[W]**
FEMIP	*GF*	696 (0.30%)	5,896 (3.31%)	7 (2.08%)	118.36	0.064
	HFE	1,106 (0.62%)	11,081 (6.22%)	6 (1.79%)	62.55	0.407
	FM	2,432 (1.36%)	656 (0.37%)	19 (5.65%)	101.30	0.037
	Total	**4,234 (2.38%)**	**17,633 (9.89%)**	**32 (9.52%)**	62.55	**2.002**

It can be noted that FEMIP occupies a limited portion of the internal FPGA resources. For this reason, FEMIP leaves enough free space for the application of the SEU-mitigation techniques, that are mandatory to use SRAM-based FPGAs in space applications(Section 3.4).

Table 5.2 compares the performances and the area occupation of FEMIP w.r.t. the actual state-of-the-art implementation (FEIC) [58] [59]. Slices and Internal memory of FEIC are reported for a *Xilinx Virtex II* device [231] (as in [59]), but the slice and memory architectures are the same as in *Xilinx Virtex 4* family devices. The reported data confirm the great improvements of FEMIP, both in terms of resources usage and speed (i.e., frames per second (*fps*)). No comparison about the power consumption has been reported, since this data is not public available for FEIC.

Table 5.2: Resource Usage for *Xilinx Virtex 4 XC4VLX200*

	Resource Usage		**Max. Speed**
	Slices	**Internal Memory [KB]**	**[fps]**
FEMIP	9,801	72	33
FEIC [59]	25,344	162.5	20
Improvements	**-61.3%**	**-55.7%**	**+65%**

To verify the correctness of the implementation, a software model of FEMIP, written in MATLAB, has been developed. The model reflects exactly the fixed-point data parallelism adopted in the hardware implementation. Results extracted from hardware simulations have been compared with the ones taken from the MATLAB simulation, and equivalence has been verified.

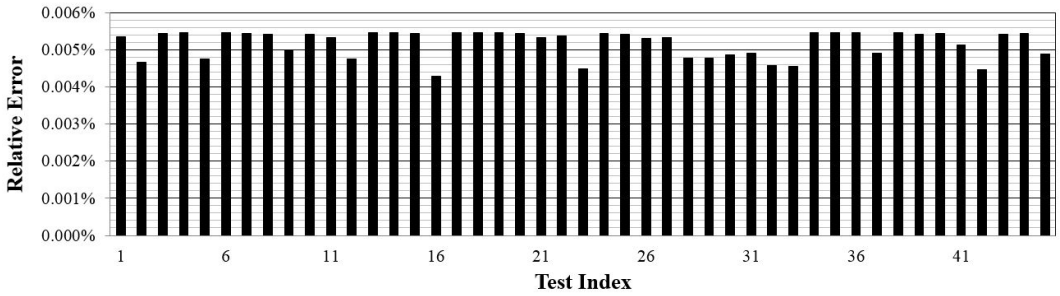


Figure 5.5: *R-factor* relative error introduced by the fixed-point representation

Moreover, in order to validate the accuracy of the proposed architecture, a second MATLAB software model of the complete FEMIP system has been implemented. It implements all the operations resorting to a double precision representation. Since the output of the Harris algorithm is thresholded in order to extract features (Section 5.1.1.2), an error is introduced only if a pixel, extracted in the MATLAB model, is not extracted in the hardware simulation, or vice versa. This misbehaviour is due to the *R-factors* fixed-point hardware approximation. Thus, the representation error has to be evaluated on values close to the threshold. The maximum relative error between the *R-factors* calculated by the *FEMIP* and those provided by MATLAB was evaluated for 45 images included in *Affine Covariant Regions Datasets* [164]. As reported in Figure 5.5, its value is always lower than 0,006%.

Moreover, by comparing the extracted points in both software and hardware simulations, we observed that, in the worst case, the representation errors led to ± 6 (out of 3,000) extracted points. Such a small difference in the set of extracted features will not decrease the effectiveness of the subsequent matching task.

Finally, in order to evaluate the overall performances of the proposed architecture, three parameters, namely *Number of Extracted Matches* (NEM), *Spatial Distribution Percentage* (SDP) and *Walsh Percentage* (WP), were evaluated (as suggested in [211]) for an images dataset (i.e., *Verification Data-set*) provided by *ThalesAlenia Space Italy S.p.a.*.

The *Number of Extracted Matches* (NEM) represents the number of extracted matches between a pair of images (i.e., a feature in the first image is identified also in the second image). While, the *Spatial Distribution Percentage* (SDP) of extracted matches is a parameter useful to quantify the distribution of the matched points on the image. To evaluate the SDP, the frame is divided into a grid of 8 by 8 cells, each of them made up of 128 by 128 pixels. In order to measures how much the extracted matching points are uniformly spread in the image, SDP is defined as:

$$SDP = \frac{\sum_{i=1}^{N} -p_i \log p_i}{\log N} \tag{5.2}$$

where p_i is computed as the number of matching points within an image cell (Section 5.1.2.2) over the total number of extracted matching points in the frame, and N is the number of image cells (i.e., 64).

Eventually, the *Walsh Percentage* (WP) is used to compare the matches extracted by the *FEMIP* with the matches found by software using the Walsh transformations (the reader may refer to [125] for more information about Walsh transform).

The higher is the value of these parameters, the more accurate will be the motion estimation task. The *Verification Data-set* is composed of 89 image pairs covering different environmental conditions (i.e. image quality), with different camera movement types, in a synthesized Mars environment (Figure 5.6). Camera movement types include displacements of 1, 10 and 30 meters at different altitudes (1,000 meters and 5,000 meters), angular rotations (0.5, 1 and 5 degrees), while image quality types include the insertion of different levels of Salt-and-Pepper noise, blur, brightness and contrast variations.

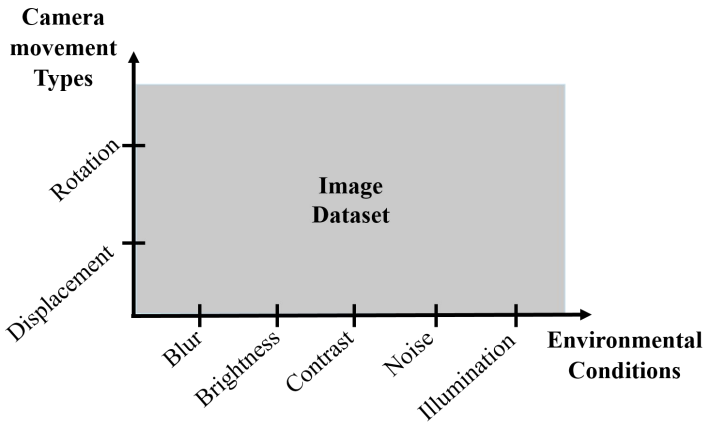
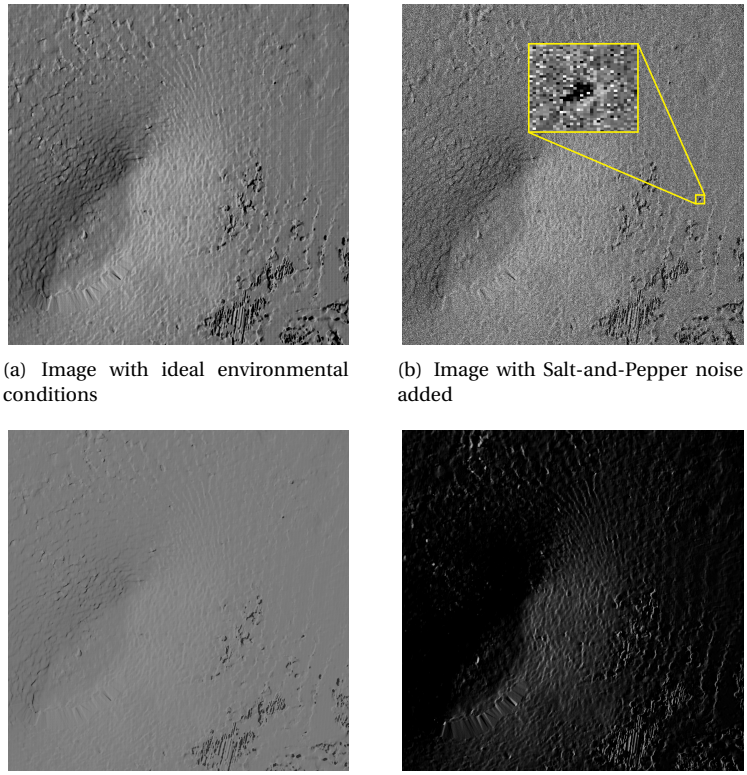


Figure 5.6: Conditions covered by the used verification image dataset

As can be seen from Figure 5.6 the test image pairs cover every combination between camera movement types and environmental conditions. Thus, tests on displacements and rotations have been repeated under different image quality conditions. Original images have been elaborated by means of software IrfanView and MATLAB in order to modify their contrast, resolution, corrupted pixels' number and brightness. The *Blur* on the image has the effect to soften the image, creating a blurry or fuzzy look. Starting from images synthesized in nominal conditions of illumination and resolution (referenced as level 0), blur has been modified using IrfanView, which enables the modification of the blur thanks to a slider control, moving between 0 to 99 (blurred image). The contrast of an image measures the difference in brightness between light and dark areas. Thus, contrast reductions can be used to simulate the fog effect. Contrast has been modified using IrfanView, which enables the modification of the number of gray levels thanks to a slider control,

moving between -127 (total grey) to 127 (black and white). For current tests, we exploited the $[-127 \div 0]$ part of the contrast control. Brightness has been also modified using IrfanView with the same method, moving between -255 (total black) to 255 (total white). For current tests, we exploited the $[-255 \div 0]$ part of the contrast control. The illumination variations simulate different Sun positions and, consequently, different shadows on the surface. Finally, Salt-and-Pepper noise has been added to the image using the MATLAB function *imnoise*. Figure 5.7 shows four examples of images included in the described dataset.



(a) Image with ideal environmental conditions

(b) Image with Salt-and-Pepper noise added

(c) Image with lower contrast

(d) Image with lower brightness

Figure 5.7: Examples of images from the Verification Dataset provided by *Thales Alenia Space Italy S.p.a.*

For the given dataset, the NEM is always greater than 200. Figure 5.8 shows the results for SDP and WP. It illustrates that the matches are almost distributed on the entire image, and a good portion of them covers the matches extracted by the *Walsh Transform*, that is considered a robust matching algorithm [125]. Furthermore, Figure 5.8 depicts that for images taken in nasty environmental conditions (i.e., indexes from 75 to 84) the WP decreases, indicating that only a subset of the extracted matches are validated also by the Walsh method, but the SDP still continue to be high.

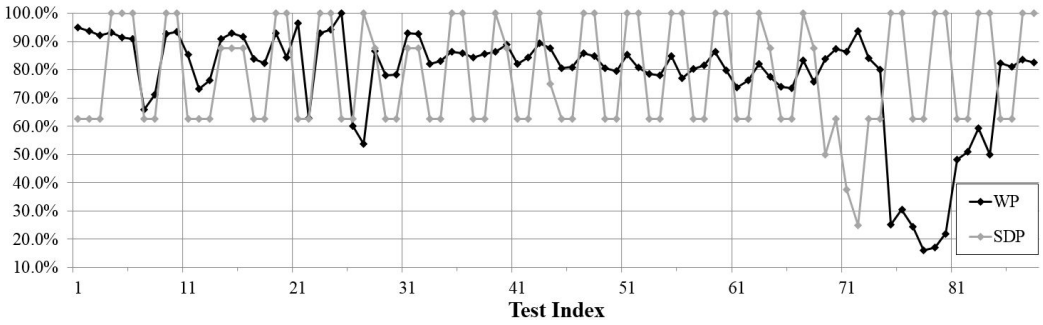


Figure 5.8: Spatial Distribution and Walsh Percentages Parameters resulting from the verification campaign

Unfortunately, a comparison of these parameters with the ones associated with FEIC (i.e., the current state-of-the-art) cannot be performed since the FEIC implementation is not public available.
For the sake of completeness, Figure 5.9 reports the matches extracted from two consecutive Mars images. The lines depict the matched features between the two images.

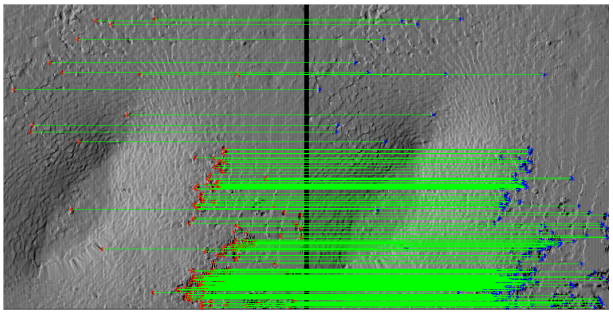


Figure 5.9: Example of extracted matches from two consecutive frames of a synthesized Mars Surface

5.1.2 SA-FEMIP

SA-FEMIP is an optimized FPGA-based self-adaptive FEM architecture based on the well known Harris feature extractor algorithm [86]. This architecture extends the architecture of FEMIP (Section 5.1.1)) by enabling self-adaptation of the parameters of the image noise filter and the feature extraction algorithm. Self-adaptation allows to better optimize the FEM algorithm to the environmental conditions, thus increasing the robustness with respect to noise and variations of external conditions that are typical of the space environment. Adaptation is obtained introducing very marginal overhead and guaranteeing high operational rates. This is achieved by resorting to the *Dynamic Partial Reconfiguration* (DPR) capabilities of modern space-qualified FPGAs.

Experimental results clearly show that SA-FEMIP enables increased accuracy and performance compared to previous architectures, two key characteristics for the implementation of VBN systems for next generation space missions.

The input and output interfaces of SA-FEMIP are the same of the ones of FEMIP. In fact, it receives a 32-bit input stream, representing a sequence of 1,024 x 1,024 grey scale frames with 10 bpp resolution, and it provides in output the set of matching points identified in the processed frames.

The SA-FEMIP pipeline (Figure 5.10) includes three main functional blocks: the *Reconfigurable Gaussian Filter*, the *Adaptive Harris Feature Extractor*, and the *Feature Matcher*. Moreover, SA-FEMIP includes an input/output interface to communicate with an external memory used to temporarily store images filtered by the *Reconfigurable Gaussian Filter* and later required during the feature matching step.

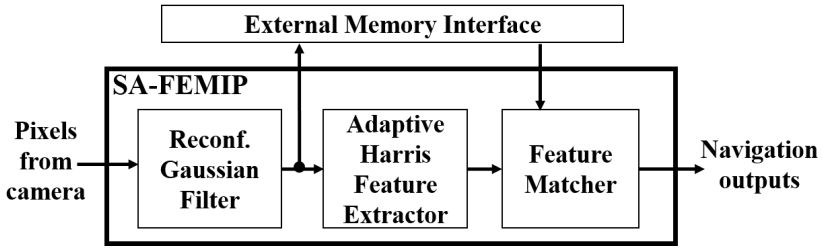


Figure 5.10: SA-FEMIP computational pipeline

The self-adaptability features of SA-FEMIP have been implemented in the *Reconfigurable Gaussian Filter* and the *Adaptive Harris Feature Extractor*, while the he *Feature Matcher* is maintained unchanged with respect to FEMIP (Section 5.1.1.3).

In the following sections a detailed description of the implemented self-adaptability features, and quantitative analysis of the obtained benefits is reported.

5.1.2.1 Reconfigurable Gaussian Filter

The *Reconfigurable Gaussian Filter* overcomes the main drawback associated with the fixed Gaussian filter variance (σ_f^2) implemented in the FEMIP Gaussian filter (Section 5.1.1.1). A fixed Gaussian filter variance works well if the noise level of the processed frames is known a priori. As an example, a high filter variance is useful for high noise levels. Instead, for low noise levels the images are over-smoothed, thus reducing the accuracy of the feature extraction and matching modules [82]. To overcome this problem, the *Reconfigurable Gaussian Filter* exploits FPGA DPR to adapt the variance σ_f^2 frame-by-frame, based on the estimated noise affecting the input frame.

In literature, many works propose adaptive filters [179][242][48][207]. Among the proposed approaches, those based on evolutionary algorithms are the most promising, in terms of timing performances and hardware resources usage [56]. Nevertheless, they provide very good results

if the processed images are similar to the one used during the training phase of the evolutionary algorithm. Instead, if the received image characteristics (e.g., illumination conditions, tonal distribution, etc.) cannot be predicted, as in the harsh space environment, the filtering performances become very poor [144] [143].
AIDI, presented in Section 4.3.1.2, proposes a Gaussian filter architecture able to self-adapt σ_f^2 pixel-by-pixel depending on the noise level affecting the processed image. This approach ensures a higher level of adaptivity with respect to evolutionary filters. However, it wastes a lot of hardware resources.

To overcome this issue, SA-FEMIP exploits FPGA DPR to provide filter adaptation while saving area and power consumption. Basically, the proposed approach estimates the level of noise affecting the input image using the same algorithm adopted in AIDI (Section 4.3.1.2). The noise level estimated for the current frame is used to select the filter variance that would guarantee optimum filtering results. This filter variance is then used to filter the next input image, allowing adaptation of the filter parameters frame-by-frame during the entire descending sequence. The adaptation of the filter variance is achieved by reconfiguring the 49 constant multipliers required to perform the convolution of the image with the Gaussian kernel (Section 5.1.1.1). This significantly saves hardware resources with respect to a solution that uses 49 generic multipliers in which the Gaussian kernel constants are selected using multiplexers driven according to the selected filter variance. Figure 5.11 shows the architecture implementing the proposed approach.

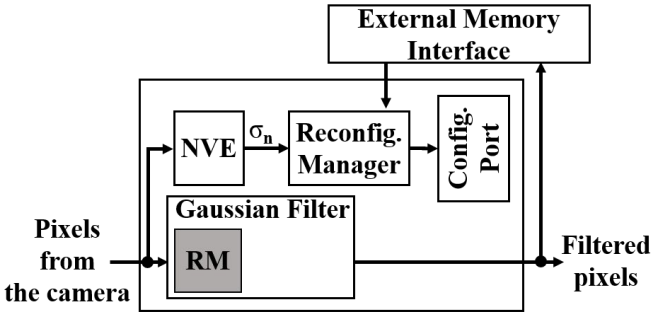


Figure 5.11: Reconfigurable Gaussian Filter hardware architecture

The *Reconfigurable Gaussian Filter* is composed of: (i) the *Noise Variance Estimator* (NVE), (ii) the *Reconfiguration Manager*, and (iii) the *Gaussian Filter*.

The *Gaussian Filter* is implemented as described in Section 5.1.1.1 but, in order to enable its reconfiguration, the 49 multipliers are enclosed in an FPGA reconfigurable module (RM in Figure 5.11). A reconfigurable module is a portion of an FPGA design that can be reconfigured at runtime, without impacting the behavior of the rest of the design.

While the *Gaussian Filter* processes the input image, the NVE estimates the noise level. The

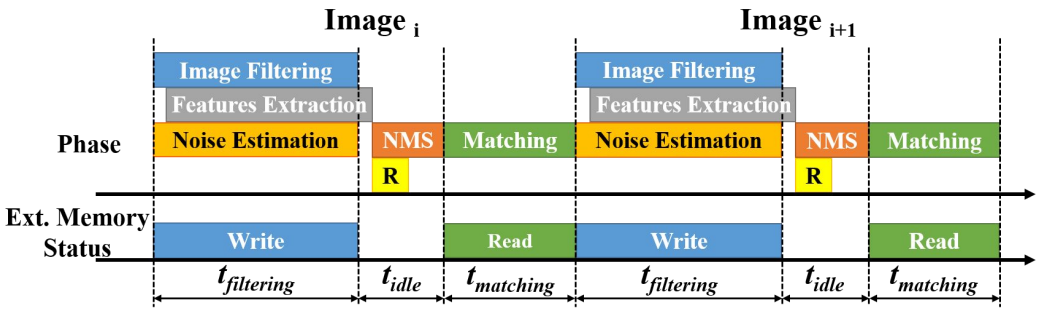


Figure 5.12: Timing diagram of SA-FEMIP

NVE is implemented adopting an architecture similar to the one presented in AIDI (Section 4.3.1.2). However, we compute here the noise standard deviation (σ_n) instead of the noise variance (σ_n^2). This change has not functional effects since σ_n is not actually used to perform calculations like in [54] during the filtering process. When a full frame has been processed, the NVE provides the current estimated σ_n to the *Reconfiguration Manager*.

The *Reconfiguration Manager* exploits this value to look-up into a bitstream address table and to select the proper configuration for the multipliers inside the RM. The multipliers reconfiguration is accomplished by reading the multipliers configuration bitstream from the external memory, choosing the configuration associated with the estimated standard deviation. The bistream is then used to program the reconfigurable module of the FPGA resorting to the FPGA *internal Configuration Port* (i.e., ICAP [230] in Xilinx FPGAs). It is worth to highlight here that using the noise standard deviation instead of the noise variance strongly reduces the NVE area, avoiding the multiplier required to compute σ_n^2.

During the reconfiguration process, the *Reconfiguration Manager* must access the external memory to retrieve the RM configuration bitstream. To avoid the stall of the processing chain, this access must be scheduled when no other module requires information from the external memory. As shown in the timing diagram of Figure 5.12, the external memory is accessed by the *Reconfigurable Gaussian Filter* in write mode to store the computed filtered pixel values. As described in Section 5.1.1.1, during this phase the *Reconfigurable Gaussian Filter* and the *Adaptive Harris Feature Extractor* work in pipeline, while the noise variance is computed (Image Filtering, Features Extraction and Noise Estimation activities in Figure 5.12). At the end of the feature extraction, the *NMS* phase takes place, and, finally, the *Feature Matcher* performs the *matching* phase where it accesses the external memory in read mode to retrieve the data needed to compute the cross-correlation. It is worth noting that the Image Filtering and the Features Extraction slots are not perfectly aligned due to the latency in loading the internal pipeline of the *Reconfigurable Gaussian Filter*. Looking at Figure 5.12, the external memory is always idle during the NMS phase (t_{idle} in Figure 5.12). This time slot can be used to reconfigure the filter (R task in Figure

5.12) without stalling the processing chain. This means that no timing overhead is introduced in the feature extraction and matching task.

Finally, since for each value of σ_f^2 a configuration bitstream must be stored in the external memory, the range of possible σ_f^2 must be discretized according to the available external memory space (Section 5.1.2.3 for detailed information about the size of each bitstream).

5.1.2.2 Adaptive Harris Feature Extractor

This section introduces the hardware architecture of the *Adaptive Harris Feature Extractor* (Figure 5.13), focusing on the novel thresholding approach that ensures to uniformly distribute the extracted features on the input frame.

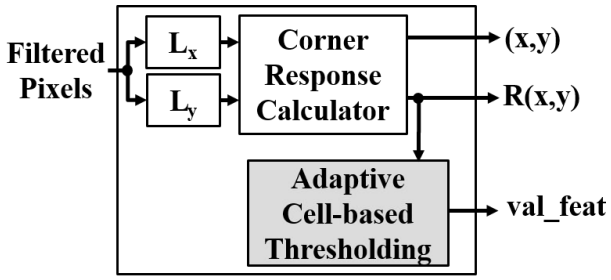


Figure 5.13: *Adaptive Harris Features Extractor* internal architecture

The first two modules of the *Adaptive Harris Feature Extractor*, L_x and L_y, compute the spatial image derivatives of the filtered image in the horizontal (L_x) and vertical (L_y) direction, respectively. This components have the same internal architecture as the namesake ones in FEMIP (Section 5.1.1). Then, the *Corner Response Calculator* module computes the determinant and the trace of the second-moment matrix $N(x,y)$, which are required to calculate the Harris corner response $R(x,y)$ associated with each input pixel. Finally, the *Adaptive Cell-based Thresholding* (ACTH) module thresholds the computed corner responses, asserting the *val_feat* signal when the current processed pixel is above the threshold and therefore represents an actual feature.

Selecting a well distributed set of features within the frame improves the motion estimation accuracy. In order to level the distribution of the extracted features on the processed frames, the ACTH module splits the input image in 64 cells of 128x128 pixels each. It then tries to extract the same number of features from each cell. This goal is achieved exploiting a local threshold for each cell, instead of using a single global threshold for the overall image, as done in FEMIP (Section 5.1.1). The chosen number of cells represents a good trade-off between accuracy and memory requirements. As it will be discussed in Section 5.1.2.3, it ensures to uniformly cover the input image and, at the same time, to avoid the introduction of a large number of cells that would require a lot of memory to store the related information items.

Algorithm 5 Adaptive Cell Thresholding approach

```
Require: NF[8,8]                                          ▷ # extracted features in each cell
Require: TH[8,8]                                          ▷ Threshold value of each cell
Require: TF[8,8]                                          ▷ # target features in each cell
Const N_cell=64                                           ▷ # of cell
Const δ=15                                                ▷ Tolerance
Const LowTH=15                                            ▷ Threshold lower bound
Const OTF=3000                                            ▷ Overall # target features
Curr_EF=∑NF[i,j]                                          ▷ Current overall # extracted features
LowTH_cell[8,8]=[0,...,0]
TF_slack=0
for i=0;i<8;i++ do
    for j=0;j<8;j++ do
        Disp=NF[i,j]-TF[i,j]
        Step = Disp * (0.5/OTF)*TH[i,j]
        if Disp< −δ then
            new_TH[i,j]=TH[i,j]+Step
            if new_TH[i,j]< LowTH then
                new_TH[i,j]=TH[i,j]
                LowTH_cell[i,j]=1
                TF_slack=TF_slack + | Disp |
            end if
        else
            if Disp> +δ then
                new_TH[i,j]=TH[i,j]+Step
            else
                new_TH[i,j]=TH[i,j]
            end if
        end if
    end for
end for
if TF_slack > 0 then
    if TF_slack < N_cell then
        TF_slack_cell = 1
    else
        TF_slack_cell = ⌊ TF_slack ÷ N_cell ⌋
    end if
    for i=0;i<8;i++ do
        for j=0;j<8;j++ do
            if Curr_EF <= OTF then
                if LowTH_cell[i,j]=0 then
                    new_TF[i,j]=TF[i,j]+TF_slack_cell
                else
                    new_TF[i,j]=NF[i,j]
                end if
            else
                if TF[i,j] = 0 then
                    new_TF[i,j]=TF[i,j]
                else
                    new_TF[i,j]=TF[i,j]-1
                end if
            end if
        end for
    end for
end if
return (new_TH, new_TF)
```

The ACTH module analyzes information related to the current frame implementing the decision process described in Algorithm 5, and computes the local thresholds to use for the following frame. The threshold adaptation process requires to know, for each cell composing the frame, (i) the number of extracted features (NF), (ii) the current threshold (TH) initialized at the highest possible value at startup (i.e., no features are extracted), and (iii) the current number of expected features (TF). In our tests, TF has been initialized to 48 to fix the overall number of expected

features per frame (OTF) to about 3,000 features. This value limits the size of the internal buffer used to store the extracted features in the *Feature Matcher* module. Since NF, TH and TF must be defined for each cell of the frame, they are stored in the form of 8x8 matrices, with the matrix elements associated with the defined frame cells.

Algorithm 5 can be split in two main parts. The former (from row 8 to 27) updates the cell threshold values. For every cell (i, j), it compares the number of extracted features $NF[i, j]$ with the number of expected features $TF[i, j]$ ($Disp$ at row 10). If these two values differ no more than a defined tolerance (i.e., the difference is contained in the range $[+\delta,-\delta]$) the threshold is not changed (row 23). Otherwise, the threshold is updated adding $Step$ to its current value (rows 13 and 21). One additional test is performed when the number of extracted features is lower than the number of expected ones (from row 14 to 18). In particular, the updated threshold ($new\_TH[i, j]$) is considered valid if it is higher than a lower bound value ($LowTH$). If not, the threshold is not changed (row 15). This avoids to over-reduce the threshold value and to provide in output weak features that could be potentially associated with the noise in the input frame. In fact, if a cell represents a flat part of the planetary surface, a high value of the image gradient, and consequently a high value of the computed corner response, is mainly due to the noise.

The second part of Algorithm 5 (from row 28 to 51) optimizes the number of features extracted for each cell in order to obtain a total number of features for the frame as close as possible OTF. To do that, it is worth to remember that all cells that reach the threshold lower bound cannot further update their threshold. If, with this threshold, the number of extracted features for the cell $NF[i, j]$ is lower than the number of expected features for the cell $TF[i, j]$ there is a certain amount of features corresponding to $|\,Disp\,|$ that can be redistributed to other cells with threshold higher than the lower bound. To exploit this, each cell with threshold lower than the lower bound is marked through the $LowTH\_cell[i, j]$ flag (row 16) and the number of unused features of these cells is accumulated in the TF_slack parameter (row 17) in order to be redistributed to the other cells, according to the decision process described from row 28 to 51. The TF_slack represents the number of expected features that can be borrowed to the cells that have not reached the threshold lower bound (i.e., $LowTH\_cell[i, j] = 0$). The number of features to borrow to each cell (TF_slack_cell) is computed dividing TF_slack by the number of cells composing the image. To ensure a high number of extracted features, the algorithm always borrows at least one feature to each cell with $LowTH\_cell[i, j] = 0$ (rows 29 to 33). If the total number of extracted features $Curr\_EF$ is lower or equal to OTF (from row 36 to 41), if the cell has not reached the threshold lower bound the number of expected features for the cell is increased of TF_slack_cell (row 38). Otherwise, it is left unchanged (row 40).

Using this approach, the total number of extracted features ($Curr\_EF$) could increase more and more due to the borrow mechanism, that increases the $TF[i, j]$ values. To allow a decrease of the $TF[i, j]$ values, and so to maintain the overall number of extracted features around OTF, if $Curr\_EF$ exceeds OTF, the target feature value of each cell is decreased by 1 (from row 43 to 47).

The hardware architecture of the ACTH module is shown in Figure 5.14.

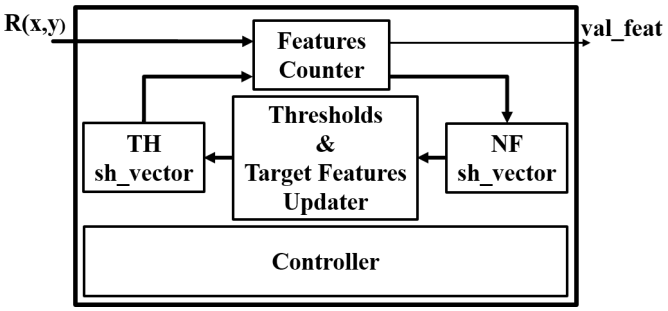


Figure 5.14: Adaptive Cell-based Thresholding hardware architecture

It is composed of four main modules: (i) the *Features Counter*, (ii) the *Thresholds & Target Features Updater*, (iii) the *TH sh_vector*, and (iv) the *NF sh_vector*.

The *Thresholds & Target Features Updater* module implements Algorithm 5, while the *Features Counter* performs the actual thresholding of each corner response $R(x,y)$ received from the *Corner Response Calculator* (Figure 5.13). This module reads the thresholds associated with each image cell (i.e., $TH[i,j]$ in Algorithm 5) that are stored in the *TH sh_vector*, and compares them with the received corner responses, asserting the *val_feat* signal if $R(x,y)$ is higher than the threshold associated with the image cell containing the currently processed pixel.

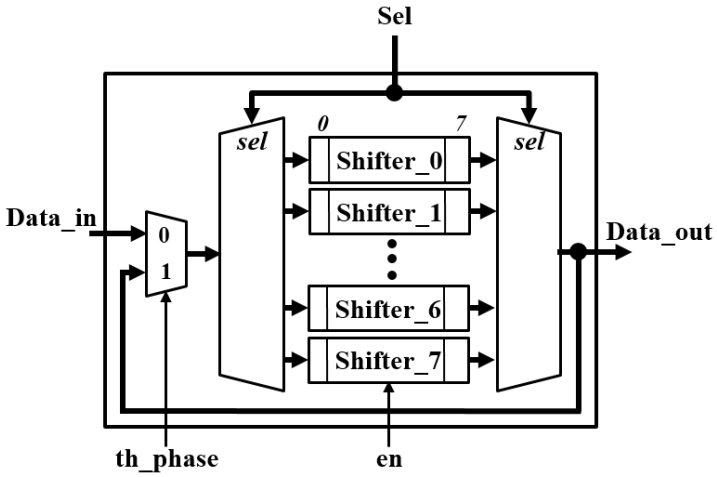


Figure 5.15: TH and NF shifter vector hardware architecture

The *TH sh_vector* module is implemented as in Figure 5.15. It is composed of eight 8-positions shift registers connected as circular buffers. Each shift register stores eight threshold values associated with a row of image cells (it is worth to remember that the image is split in 64 cells orga-

nized in 8 rows with 8 cells each, and a threshold value is associated with each cell). The *en* signal enables the 1-position right shifting operation, while the *Sel* signal selects which shift register must be rotated. These two control signals are driven in order to provide in output the threshold associated with the image cell of the currently processed pixel. Since the image is received in a raster way, and each image cell is composed of 128x128 pixels, *en* is asserted for a clock cycle every 128 received corner responses (i.e., whenever we move from a cell to the following one). Instead, *Sel* selects the next shift register (i.e., the next row of image cells) after 128x1024 received corner responses (i.e., whenever a complete row of image cells has been processed). To avoid loosing the threshold values, during the thresholding phase each shift register composing the *TH sh_vector* acts as a circular buffer through the multiplexer driven by the *th_phase* signal (Figure 5.14). Instead, during the thresholds updating phase, the content of the *TH sh_vector* is overwritten (exploiting the *Data_in* port) with new thresholds values computed by the *Thresholds & Target Features Updater* module.

Simultaneously to the thresholding task, the *Features Counter* counts (through an accumulator) the number of extracted features for each image cell (i.e., $NF[i,j]$ in Algorithm 5), and stores these values inside the *NF sh_vector*. The *NF sh_vector* is implemented as the *TH sh_vector* (Figure 5.15), and both modules share the input control signals. Whenever we move from the current image cell to the next one, the content of the internal accumulator is stored inside the *NF sh_vector*, and it is initialized with the output value provided by the *NF sh_vector*. At the end of the operations described by Algorithm 5, a local reset is asserted to clear the content of the *NF sh_vector* in order to prepare it for the next image processing cycle. All aforementioned control signals are generated by the *Controller* module (Figure 5.14), which also coordinates the operations of all modules included in the *ACTH*.

5.1.2.3 Experimental Results

To estimate the hardware resources and the timing performances, the SA-FEMIP architecture has been synthesized and implemented, resorting to *Xilinx ISE Design Suite 14.6* [229], on a *Xilinx Virtex 4 XC4VLX200* [226]. The reasons for the selection of this device are the same of the ones reported in Section 5.1.1.4. Post place and route simulations have been done using *Modelsim SE 10.1c* [83] to annotate the switching activities of internal nodes, and the *Xilinx XPower Analyzer* has been exploited of power consumption estimation.

Table 5.3 compares the SA-FEMIP adaptive architecture with the FEMIP one (Section 5.1.1). Comparison is performed in terms of area overhead by considering internal logic and memory resources (i.e., registers, Look-Up Tables (LUTs), and Block-RAMs (BRAMs) (Section 3.3)). Percentages in Table 5.3 represent the used portion of the hardware resources available in the *Xilinx Virtex 4 XC4VLX200* [226]. It is important to point out that the synthesis of both *FEMIP* and *SA-FEMIP* architectures has been forced to avoid the use of DSPs. The reasons for this choice will be

better elaborated later in this section. Power consumption is analysed considering an operating frequency of 60 MHz for both architectures.

Table 5.3 shows that SA-FEMIP FPGA occupation is around 10% for logic and memory resources,

Table 5.3: Resources usage and power consumption of FEMIP and SA-FEMIP, implemented on a *Xilinx Virtex 4 XC4VLX200*

	Module	FPGA Area Occupation			Max.Freq.	Power
		Registers	LUTs	BRAMs	[MHz]	[W]
FEMIP	*GF*	696 (0.30%)	5,896 (3.31%)	7 (2.08%)	118.36	0.064
	HFE	1,106 (0.62%)	11,081 (6.22%)	6 (1.79%)	62.55	0.407
	FM	2,432 (1.36%)	656 (0.37%)	19 (5.65%)	101.30	0.037
	Total	**4,234 (2.38%)**	**17,633 (9.89%)**	**32 (9.52%)**	62.55	**2.002**
SA-FEMIP	*RF*	939 (0.53%)	7,448 (4.18%)	10 (2.98%)	118.36	0.083
	AHFE	1,362 (0.76%)	12,468 (7.00%)	6 (1.79%)	62.55	0.462
	FM	2,432 (1.36%)	656 (0.37%)	19 (5.65%)	101.30	0.037
	Total	**4,733 (2.66%)**	**20,576 (11.55%)**	**35 (10.42%)**	62.55	**2.097**
Overhead	**Total**	**499 (0.28%)**	**2,943 (1.66%)**	**3 (0.90%)**	0	**0.095**

and the overhead w.r.t. FEMIP is less than 2%. This overhead is due to the additional modules required to perform adaptation in the *Reconfigurable Gaussian Filter* (RF) described in Section 5.1.2.1 and the additional hardware required to implement the *Adaptive Harris Features Extractor* (AHFE) presented in Section 5.1.2.2. In particular, in the RF, the increased occupation is due to the NVE and the Reconfiguration Manager modules. Instead, in the AHFE, the area overhead is introduced by the usage of a more complex thresholding approach, with respect to the simple one adopted in FEMIP. It is worth to highlight here that an effort has been placed to limit the registers overhead. The AHFE architecture strongly relies on shift registers structures to implement the required vectors and matrices included in Algorithm 5. This kind of component can be efficiently implemented in Xilinx FPGAs, exploiting the Xilinx *SRL* capability of the Look-Up Tables (LUTs) (Section 3.3). This capability makes it possible to use LUTs as shift registers instead of a chain of Flip-Flops, saving hardware resources. As an example, a single LUT, in a *Xilinx Virtex 4*, can act as a 16x1-bit shift register avoiding a chain of 16 flip-flops.

The maximum operating frequencies of each module reported in Table 5.3 demonstrate that no timing penalty is introduced in SA-FEMIP by the introduction of the adaptivity features.

The power consumption of each module reported in Table 5.3 does not take into account the contribution of the clock circuitry and the leakage. These contributions are included in the overall power consumption. By comparing the power consumption of SA-FEMIP with the one of FEMIP a very limited overhead equal to 4.75% is observed. It is worth noting that the power consumption of the RF module does not include the power used during the partial reconfiguration

process. However, according to [21] the reconfiguration process consumes few tens of mW, only.

Eventually, the throughput, in terms of frames-per-second (fps), is the same (i.e., 33 fps) for both FEMIP and SA-FEMIP.

In Table 5.4, the performances and the area occupation of SA-FEMIP have been compared with FEIC [58] [59]. FEIC is a Feature Extraction and matching Integrated Circuit, based on the Harris algorithm, that University of Dundee developed for the European Space Agency (ESA) in the framework of the Navigation for Planetary Approach and Landing (NPAL) project. LUTs and BRAMs used by FEIC are reported for a *Xilinx Virtex II* device [231] (as in [59]), but the internal logic and memory architecture is the same as in *Xilinx Virtex 4* family devices. The reported data confirm the great improvements of the SA-FEMIP architecture, both in terms of resources usage and throughput.

Table 5.4: Resource usage and throughput of *FEIC* and SA-FEMIP for a *Xilinx Virtex 4 XC4VLX200*

	Resource Usage		**Max. Speed**
	LUTs	**BRAMs [KB]**	**[fps]**
SA-FEMIP	20,576	78.75	33
FEIC [59]	50,688	162.5	20
Improvements	**-59.4%**	**-51.5%**	**+65%**

The low area occupation of SA-FEMIP allow designers to exploit the free hardware resources to apply fault mitigation strategies to increase the reliability of the design, a key requirement in space applications (Section 3.4). Even after the implementation of fault tolerance techniques, space is also available to integrate in the same device additional FPGA-based IP-cores useful to accelerate other computational intensive tasks performed during the descending phase (e.g., *Hazard map computation* [214]). This is very important considering the limited resources available in space applications.

As mentioned at the beginning of this section SA-FEMIP has been synthesized avoiding the use of DSPs. This decision can now be better motivated. DSPs have the advantage of further reducing the area occupation of FEMIP especially when multipliers are implemented. With the use of DSPs the SA-FEMIP occupation would be reduced to 9,029 (5.06%) LUTs, 66 (68.75%) DSPs, while the occupation of registers and BRAMs remains the same. Nevertheless, DSPs are limited resources. With 66 DSPs required out of the 96 available in the *Xilinx Virtex 4-QV VLX200* [226], TMR techniques for this portion of the design would not be possible. Moreover, the intensive use of DSPs increase the routing complexity introducing a 30% frequency penalty in the design.
SA-FEMIP has not been compared to [26], since [26] implements the multi-scale Harris detector (i.e., a rotation-invariant version of the Harris detector). [26] consumes a lot of hardware resources, and implements a feature that is not actually required in EDL applications since rotations between two consecutive images are limited [63].

SA-FEMIP has been evaluated in terms of accuracy and robustness, exploiting an image dataset, provided by *Thales Alenia Space Italia s.p.a.* company, that covers different landing zones (i.e., portions of the Mars surface), environmental conditions (i.e., image quality), and camera movement types, in a synthesized Mars environment. Camera movement types include displacements, up to 30 meters, at different altitudes (from 1,000 meters to 5,000 meters), and angular speed (up to 2.5 $^{\circ}/s$, in accordance to [63]), while image quality types include the injection of different levels of Gaussian noise, blur, brightness and contrast variations. This dataset is an improved version of the *Verification Data-set* presented in Section 5.1.1.4, that contains image sequence representing actual descending sequences on the Mars surface.
According to [63], the robustness has been evaluated exploiting two parameters: (i) *Number of Extracted Matches* (NEM), and (ii) *Spatial Distribution of Points* (SDP) (Section 5.1.1.4).

Figure 5.16 shows the SDP results obtained from FEMIP and SA-FEMIP by providing in input the images composing the aforementioned dataset. Thanks to the adaptive cell-based thresholding approach, SA-FEMIP outperforms FEMIP results in every test case (i.e., Test Index). In particular, the improvements are very high (from Test Index 0 to 76) when the input images represent a landing zone characterized by few small rugged regions. This is visually highlighted in Figure 5.17 that depicts the matching points extracted by FEMIP (Figure 5.17(a)) and SA-FEMIP (Figure 5.17(b)). Each figure shows two consecutive input images with lines connecting the features that match in the two images.

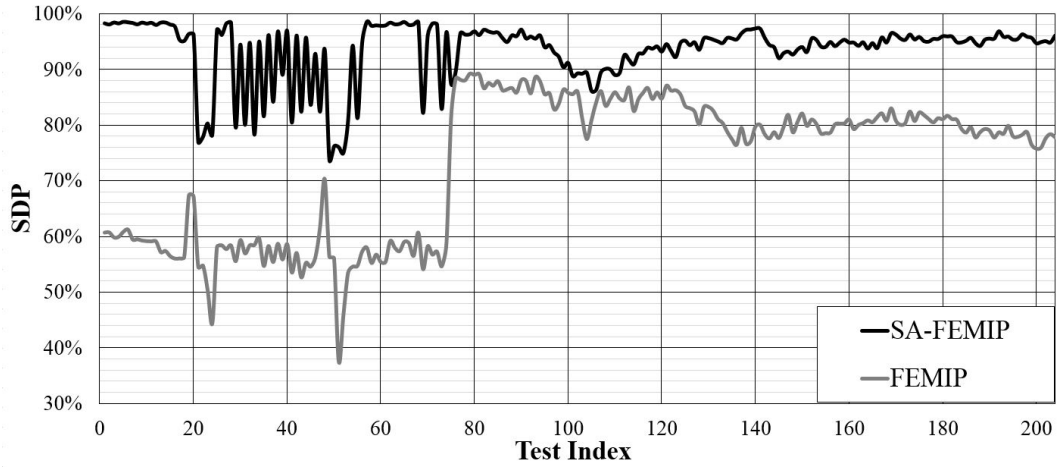
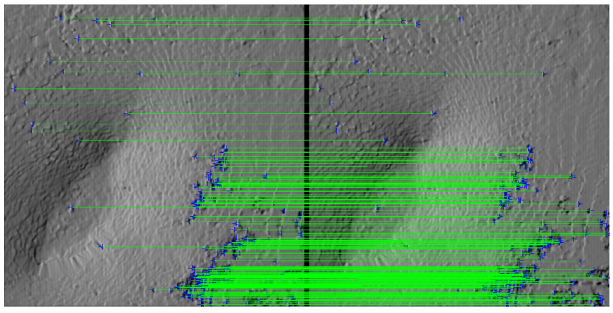
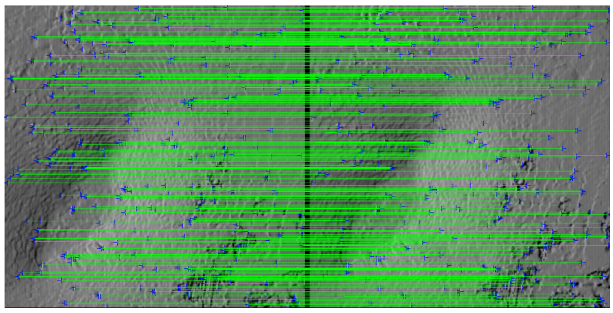


Figure 5.16: SDP results for FEMIP and SA-FEMIP

Figure 5.18 shows the NEM versus different levels of injected Gaussian noise variance σ_f^2 (since FEMIP has a fixed $\sigma_f^2 = 2$, its NEM is represented by the dashed line).
A fixed σ_f^2 does not allow to reach the highest NEM for every noise level. Thus, exploiting the reconfigurable filter architecture (Section 5.1.2.1) it is possible to highly increase the number of



(a) FEMIP



(b) SA-FEMIP

Figure 5.17: Example of extracted matches

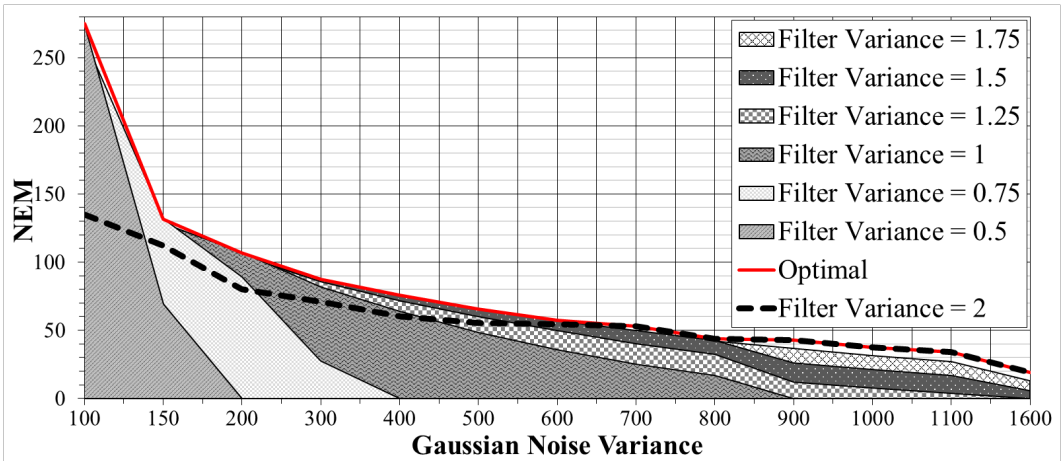


Figure 5.18: NEM results for different levels of injected Gaussian noise, varying the Gaussian Filter variance

extracted matches, as shown by the *Optimal* line in Figure 5.18. In order to follow the trend of this line, in the SA-FEMIP architecture 5 configurations for the RF module have been chosen. In particular, these configurations are associated with σ_f^2 equal to 0.5, 0.75, 1, 1.5 and 2, for the

noise level ranges [0,100], [100,200], [200,300], [300,600], [600,1600], respectively. As can be seen in Figure 5.18, the usage of a reconfigurable filter increases the NEM value w.r.t. FEMIP up to 2 times, especially for a σ_n^2 lower than 600.

Moreover, as described in Section 5.1.2.1, the usage of the DPR enables to save resources with respect to use a static hardware architecture including 49 multipliers, each one with a multiplexer to select the right Gaussian kernel value. In SA-FEMIP, using the same fixed-point data representation adopted in FEMIP, the RM and the *Reconfiguration Controller* (Figure 5.11) require 5,320 LUTs and few registers. Instead, a static hardware architecture (as the one adopted in AIDI), with the same data parallelism, would require about 19,000 LUTs, leading to a save of 72% of hardware resources.

Since each bitstream for the RM module is 166 KB (for the selected FPGA device), to store the 5 configurations 830 KB are required in the external memory. Since the throughput of the *Reconfiguration Controller* is 400 MB/s (i.e., this value is limited by the maximum throughput of the *ICAP* [230]), the time required to reconfigure the RM is equal to 0.42 ms. This time fits the idle time of the external memory (i.e., t_{idle} in Figure 5.12) that is equal to 1.2 ms (i.e., the time required by the *Matcher* to perform the *NMS* phase). For the sake of completeness, considering an operating frequency of SA-FEMIP chain equal to 60 MHz, the time required to perform the filtering and the matching tasks (i.e., $t_{filtering}$ and $t_{matching}$ in Figure 5.12) is 21.5 ms and 7.6 ms, respectively.

Eventually, Figure 5.19 shows the percentages of Correct Matches (CM) for the different filter configurations and injected noise levels. CM has been computed exploiting the knowledge about

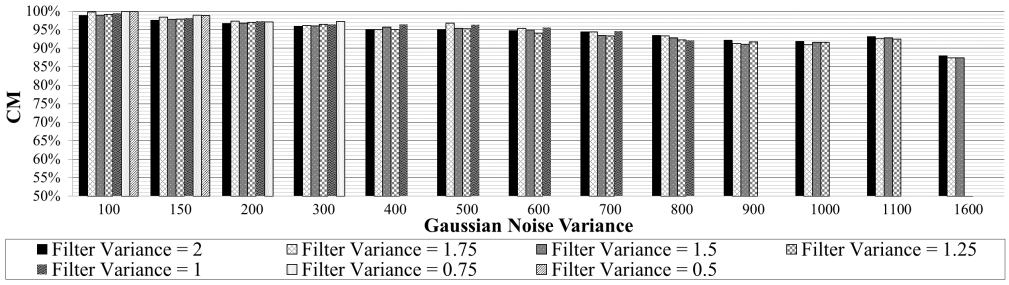


Figure 5.19: Correct Matches (CM) results for different levels of injected Gaussian noise, varying the Gaussian Filter variance

the camera movement between two consecutive images of the dataset. Starting from the position of a matching point in the first image, it is possible to compute its expected position in the second image by using a three dimensional roto-translation model. For each couple of images in the dataset this process has been automated through a *MATLAB* script. Then, the CM values have been computed by comparing the outputs of the script with the ones of the proposed architecture.

It is worth noting that, the CM values are not computed for every σ_f^2 since, as shown in Figure

5.18, with a filter characterized by a low variance it is not possible to extract matching points for very high noise levels.
As can be seen in Figure 5.19, the accuracy of the different filter configurations is higher than 85% for every noise level, and it is almost equal for a fixed noise level. These data demonstrate that the proposed filter is able to maximize the NEM, while preserving the correctness of its outputs.

5.2 FPGA-based hardware accelerators for Active Space Debris removal

The system presented in this section takes care of accelerating the first phase of the debris removal mission. In order to collect the required information about the object to be removed three main operations must be performed: (i) the debris three-dimensional shape reconstruction, (ii) the definition of the structure of the object to be removed and the identification of the composing material, and (iii) the computation of the kinematic model of the debris. In particular this work focuses on the first of these three phases.

As described in Section 2.6, to enable the debris three-dimensional shape reconstruction, video-based techniques are the most suitable in the strict context of space missions, where low-energy consumption is fundamental, and sensors should be passive in order to avoid any possible damage to external objects as well as to the chaser satellite.

Digital cameras can be used for 3D shape reconstruction exploiting several techniques. In particular, as highlighted in Sections 2.6.1 and 2.6.2, the most used and efficient techniques are the stereo-vision, and Shape-from-shading approaches.
However, both methods cannot provide the expected performances on every kind of objects to be reconstructed. In fact, stereo-vision technique limitations arise when the object to be captured is mono-chromatic and texture-less [197]. Since several types of space debris fall into this category, Shape-from-Shading algorithms, that work exploiting pixel intensities in a single image [238] could represent an efficient solution.

Moreover, in this type of missions, real-time, or quasi-real-time performances are required to ensure high accuracy of the reconstructed shape models. Since image processing algorithms are very computational intensive, a software implementation of these algorithms running on a modern fault tolerant space-qualified processor (e.g., LEON3 [75]) cannot achieve the required performances. In this context hardware acceleration is crucial and devices such as FPGAs best fit the demand of computational capabilities.

For all the aforementioned reasons, this chapter presents a novel fast shape-from-shading (SfS) algorithm and a field-programmable gate array (FPGA)-based system hardware architecture for video-based shape reconstruction of space debris. The FPGA-based architecture, equipped with a pair of cameras, includes a fast image pre-processing module, a core implementing a feature-based stereo-vision approach, and an on-board processor that executes the novel SfS

algorithm. The proposed combination of fast hardware accelerators and optimized software routines ensure to maintain limited the hardware resources usage, while in the same time provide high processing speed. The comparisons done with other state-of-the-art SfS methods demonstrates the provided timing improvements and lowest area occupation. In particular, the reduction in the hardware resources usage ensures to integrate in a single FPGA device other vision-based techniques to improve the comprehension of debris model, allowing a fast evaluation of associated kinematics in order to select the most appropriate approach for capture of the target space debris (i.e., another key step in the active space debris removal (Section 2.6)).

The main novelty of the proposed system concerns the overcoming of existing algorithm limitations. All the SfS algorithms analysed in Section 2.6.2 present three main limitations: (i) they are very far from real-time behaviours, (ii) their outputs represent the normalized shape of the observed object with respect to the brightness range in the input image, without providing information on its absolute size and absolute distance from the observer, and (iii) they create artefacts if the surface is not completely monochromatic. Thanks to the proposed system, from one hand, the first problem is solved by the novel fast Shape-from-Shading algorithm which exploits the knowledge on the input light direction (that is easily retrievable during space missions), with respect to the image plane, to reduce the computational load of the shape reconstruction problem. On the other hand, the last two problems can be solved by merging stereo-vision and shape-from-shading approaches. In particular, depth data can be exploited to correct and de-normalize the shape extracted by the SfS algorithm, in order to increase the robustness of the entire shape reconstruction process.

In the next sections, the novel fast Shape-from-Shading algorithm, and the proposed system hardware architecture are detailed. Eventually, the chapter is concluded with the presentation of the experimental results highlighting the benefits of the proposed system w.r.t. the state-of-the-art solutions.

5.2.1 Novel Fast Shape-from-Shading Algorithm

This section describes the proposed novel fast Shape-from-Shading algorithm, hereafter called *Fast-SfS*.

The basic idea of *Fast-SfS* is to suppose, as for the regular SfS algorithms, that the surface of the captured object follows the rules of the Lambertian model [238]. However, a first main differentiation from the standard SfS algorithms, concerning the light direction, is introduced. Considering the specific Space Debris removal application, the dominant source that lights the observed object is the Sun. In fact, the Albedo of the earth (i.e., the percentage of the sun light reflected by the earth surface and atmosphere back to the space) can vary from the 0% (e.g., ocean, sea, and bays) to 30% (e.g., clouds) [162]. Thus, also in the worst case, the sun represents the dominant light source. Since data concerning the actual environmental conditions in space debris removal

missions are still not available, in this work we initially assume that the earth reflection is negligible. Taking into account the data about the earth Albedo, this assumption can be considered valid, indeed the elements that provide the lower reflection factor (e.g., ocean, sea, and bays) are also the ones that cover the major part of the earth surface (i.e., around the 71% of the overall earth surface[205]). Knowing the position and the orientation of the system which captures the images with respect to the sun, it is possible to determine the mutual direction between the cameras axis and the sunlight direction. This information can be extracted by computing the attitude of the spacecraft, which is provided by Sun-sensors [224] or Star-trackers [137], that are commonly available on spacecrafts.

A second assumption is about the light properties, which is supposed to be perfectly diffused, since the sun is far enough to be considered as an illumination source situated at infinite distance. This means that sunlight rays can be considered parallel among each other.

Given the two aforementioned assumptions, the amount of light reflected by one point in the observed object surface will be proportional to the angle between the normal (n) to the tangent surface in this point, and the light direction. Thus, all points in which n is parallel to the light direction (as (1) in Figure 5.20) are supposed to be represented in the image as the ones with maximum brightness (i.e., these pixels provide the maximum reflection). On the contrary, all points in which n is perpendicular to the light direction (as (3a) and (3b) in Figure 5.20) are represented with the minimum image brightness (i.e., these pixels do not provide reflection).

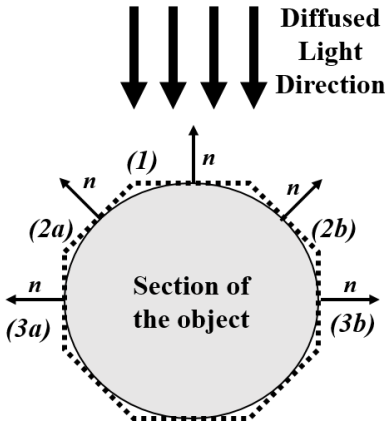


Figure 5.20: Example of the light reflected by the object surface, supposing diffused light.

The proposed algorithm can be summarized by the flow diagram in Figure 5.21.
First, the algorithm crops the input image following the object borders, in order to exclude the background pixels from the following computations. Cropping is performed by means of a single thresholding algorithm that, for each row of the image, finds the first and the last pixel in that row that are greater than a given threshold, representing the background level. Moreover, *Fast-SfS*

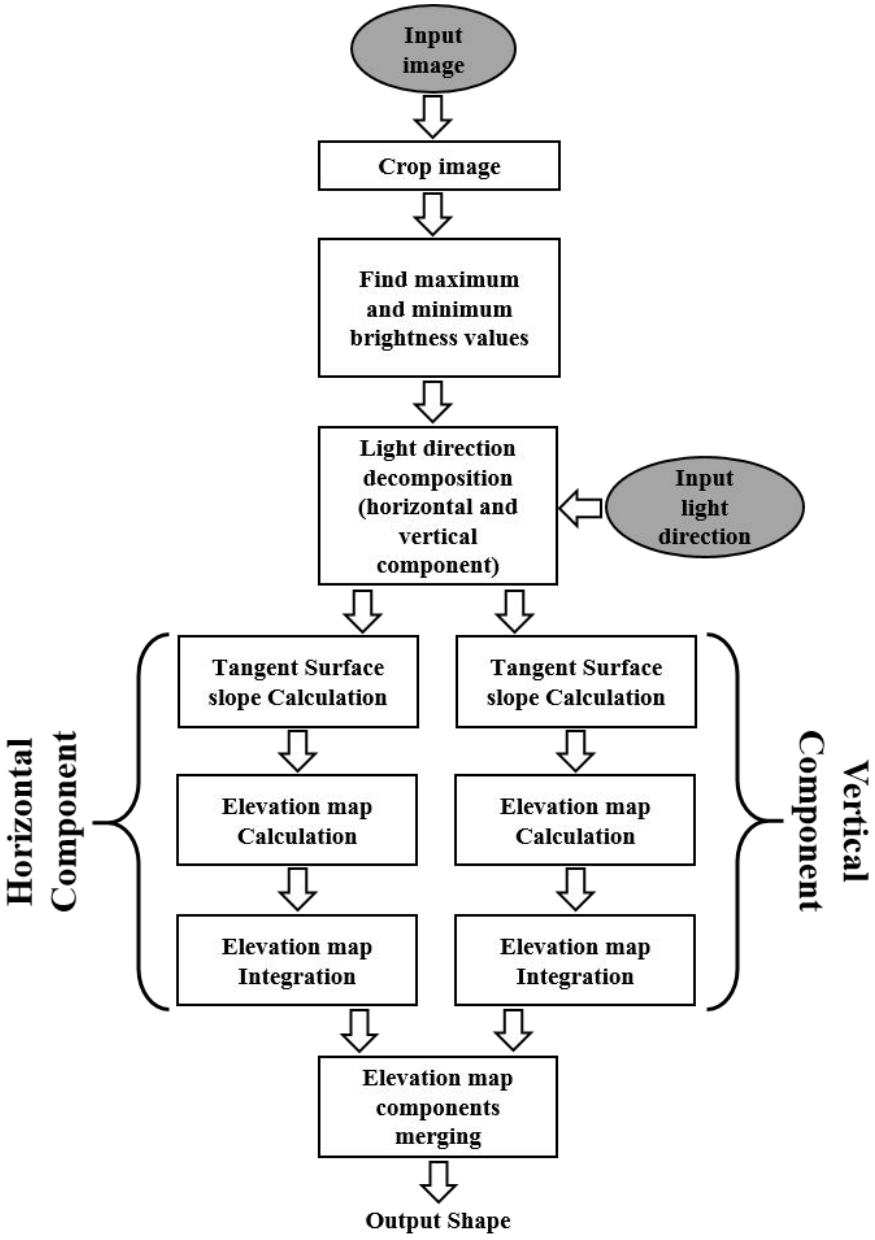
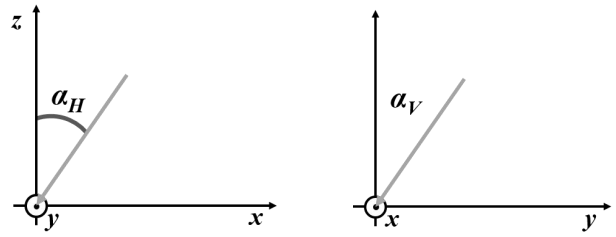


Figure 5.21: Flow diagram of the proposed *Fast-SfS* algorithm.

searches for the minimum and the maximum brightness values inside the cropped image.
Then, the proposed algorithm computes the position in the 3D space of each pixel composing the cropped image. Considering the image surface as the *x*-*y* plane, and the *z* axis as the normal to the image plane, the position of a pixel in the 3D space is defined exploiting the associated *(x, y, z)*

coordinates. Obviously the first two coordinates (i.e., x and y) are simply provided by the position of the pixel in the image surface. Instead, the elevation of each pixel (i.e., z coordinate) is defined exploiting the light direction, that is provided in input as the horizontal and vertical components. These components are provided as the two angles between the axis z and the horizontal (i.e., the projection on the x-z plane) and vertical projections (i.e., the projection on the y-z plane) of the vector representing the light direction. For a better comprehension, Figure 5.22 shows the two components of the light direction, where α_H is the component along the horizontal axis, and α_V is the one along the vertical axis.



(a) Component on image horizontal axis

(b) Component on image vertical axis

Figure 5.22: Light direction decomposition

As shown in Figure 5.21, the elevation of each pixel is separately computed along the horizontal and vertical component, that are finally merged together to compute the shape associate to the input image. This approach ensures the reduction of complexity of the operations to be performed and potentially allows to parallelize the computations. For the sake of brevity, in the following, the algorithm details are reported for the computations related to the horizontal direction only. However, the same considerations are valid for the vertical components.

The elevation of each pixel is computed in two steps according to the reflection model previously described (Figure 5.20).
First, for each pixel, its intensity value is exploited to find the slope of the tangent surface to the object point represented by the pixel. According to the considered light reflection model, Figure 5.23 shows the case in which the currently processed pixel (*P(x,y)*) is characterized by the maximum brightness value in the image. In this case, the tangent surface associated with *P(x,y)* (*S(P(x,y))*) is perpendicular to the light direction component (i.e., like surface *(1)* in Figure 5.20), so exploiting the similar triangle theorem, it is possible to demonstrate that the slope (i,e., the angle between *S(P(x,y))* and the x axis, called $\alpha_{P(x,y)}$) is equal to α_H.
In the opposite case, when the considered pixel presents the minimum brightness value, the tangent surface is parallel to the light direction (Figure 5.24), and $\alpha_{P(x,y)}$ is equal to $90° - \alpha_H$.
Considering that the pixels with the maximum brightness (I_{max}) have an associated angle equal to α_H, and the pixels characterized by the minimum value of brightness (I_{min}) have an associ-

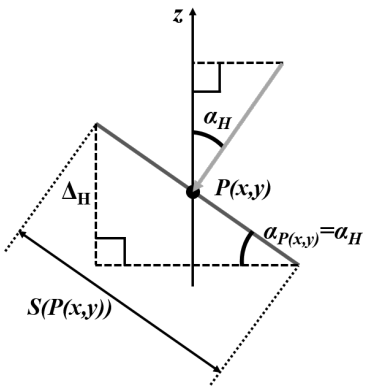


Figure 5.23: Case in which $P(x, y)$ has the maximum intensity value.

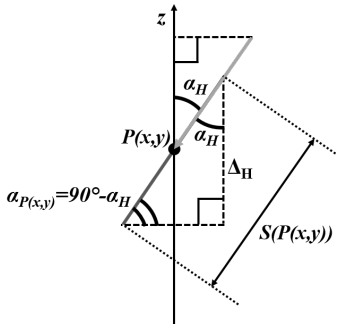


Figure 5.24: Case in which $P(x, y)$ has the minimum intensity value.

ated angle equal to $90° - \alpha_H$, an $\alpha_{P(x,y)}$ value can be assigned to all other pixels in the image by linearly regressing the range $[\alpha_H; 90° - \alpha_H]$ on the pixel brightness range. Figure 5.25 shows the proposed linear regression model.

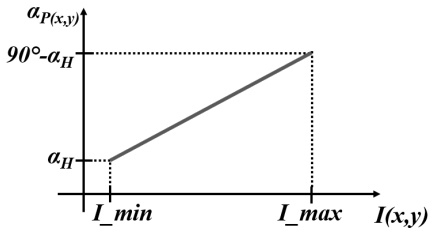


Figure 5.25: Proposed linear regression model

According to the graph in Figure 5.25, the $\alpha_{P(x,y)}$ value can be computed for each pixel as:

$$\alpha_{P(x,y)} = \frac{90° - 2 * \alpha_H}{I_{max} - I_{min}} * I(x, y) + q$$

where $I_{x,y}$ is the brightness value of the current pixel, and q is equal to:

$$q = \frac{\alpha_H(I_{max} + I_{min}) - 90^\circ * I_{min}}{I_{max} - I_{min}}$$

Afterwards, in order to extract the elevation map of each pixel, i.e., Δ_H in Figure 5.23 and Figure 5.24, the tangent of each $\alpha_{P(x,y)}$ is computed. In this step, only the module of the resulting tangent value is taken into account, since from the brightness of a pixel it is not possible to define the sign of the slope (as surfaces *(2a)* and *(2b)* in Figure 5.20).

Finally, the Δ_H are merged together to create the complete elevation map (or the object profile) associated with the horizontal component. This is done by integrating pixel-by-pixel the Δ_H values. Thus, the final elevation of each pixel is the sum of all Δ_H associated with the pixels that precede it in the current image row. This operation is repeated for each row of the cropped image. To discriminate if the object profile decreases or increases (i.e., if Δ_H is positive or negative), the brightness value of the currently considered pixel $I(x,y)$, is compared with the one of the previous in the row $I(x-1,y)$. If $I(x,y) \geq I(x-1,y)$ the Δ_H is considered positive (i.e., the profile of the object increases), otherwise it is negative.

As shown in the flow diagram of Figure 5.21 all the aforementioned operations are repeated for the vertical component.

Finally, the two elevation map components are merged to obtain the computed shape results. The two components are combined using the following equation:

$$H(i,j) = \sqrt{\Delta_{H_x}(i,j)^2 + \Delta_{H_y}(i,j)^2}$$

where $H(i,j)$ represents the output elevation map matrix, Δ_{H_x} and Δ_{H_y} are the two components of the elevation map in the horizontal and vertical axis, respectively, while (i,j) represents the pixel position in the image.

The most complex operation that the algorithm must perform is represented by the tangent computation. However, to allow a fast execution, this function can be approximated using a Look-Up Table approach. Moreover, compared to the SfS algorithms introduced in the previous section, *Fast-SfS* is not iterative, it does not present any minimization function, and can be parallelized.

Obviously, *Fast-SfS* presents the same problems, in terms of output results, as all the other SfS algorithms that rely on the Lambertian surface model. Nonetheless, these problems can be overcome resorting to stereo-vision depth measures [35, 176].

5.2.2 Proposed hardware architecture

The overall architecture of the proposed system is shown in Figure 5.26. It is mainly composed of the *FPGA subsystem* and the *Processor subsystem*.

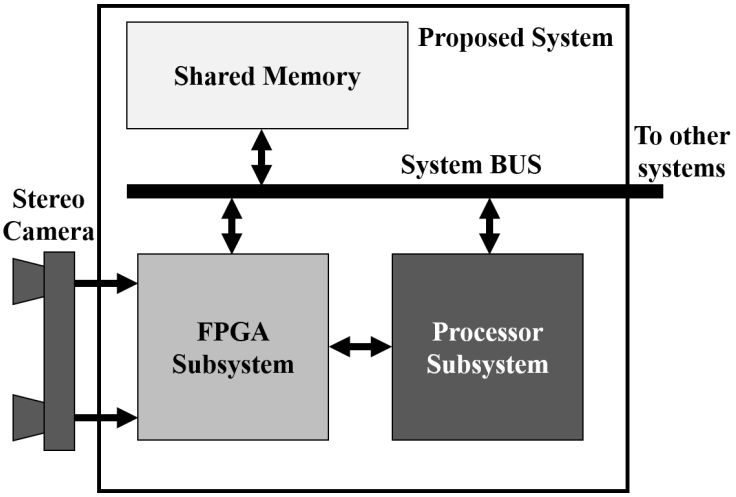


Figure 5.26: Proposed architecture.

The *Stereovision Camera* provides in output two 1,024 x 1,024 gray-scale images, with 8 bit-per-pixel resolution. The camera is directly connected to the *FPGA sub-system* (i.e., an FPGA device), that is in charge of acquiring the two images at the same time, and pre-processing them in order to enhance their quality. Moreover, the FPGA also implements a feature-based matching algorithm to provide a "sparse" depth map of the observed object (i.e., it provides depth information of extracted features, only).

In parallel to the feature-based matching algorithm, the *Processor sub-system* performs the novel fast Shape-from-Shading algorithm, presented in the previous section, and provides in output the reconstructed shape of the actual observed object portion.

The *FPGA* and the *Processor subsystems* share an external memory, used to store results and temporary data, and they communicate between each other in order to avoid any collision during memory reads or writes. As aforementioned, the results obtained by the stereo-vision algorithm can be merged to the ones in output from the Fast-SfS algorithm to correct them and to enhance their accuracy [35, 176].

The following subsections details the functions and the internal architecture of the two sub-systems.

5.2.2.1 FPGA sub-system

As depicted in Figure 5.27, the *FPGA subsystem* is composed of an FPGA device which includes several hardware-implemented modules.

Its internal architecture can be split in two main stages. The first, called *Image Preprocessing Stage*, encloses the *Input Controller*, the *Noise Filters*, the *Image Enhancers*, and the *Rectifiers*, while the second, called *Stereo-vision Processing Stage*, includes the *Feature Extractors*, the *Fea-*

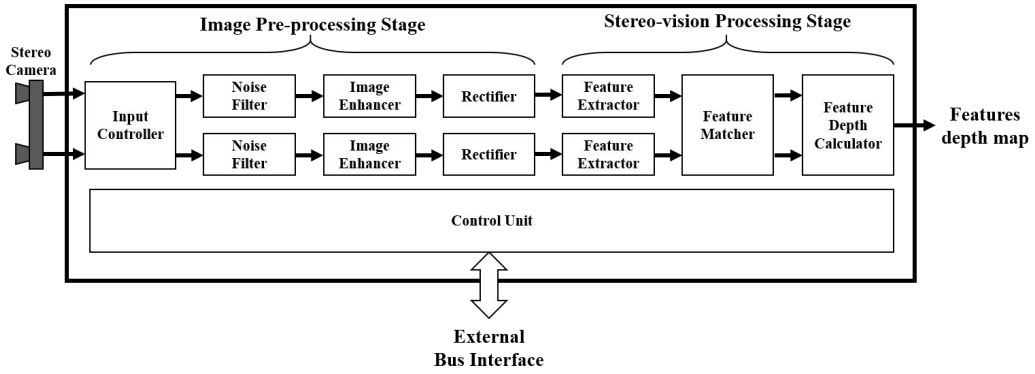


Figure 5.27: FPGA subsystem internal architecture.

ture Matcher, and the *Feature Depth Calculator*. Moreover, a main *Control Unit* coordinates all the activities of the different modules, providing also the interface with the external bus, to communicate with the *Shared Memory*.

An FPGA-based hardware implementation of these algorithms has been preferred with respect to a software implementation running on an embedded processor since, when dealing with 1,024 x 1,024 pixels images, the software alternative can lead to execution times in the order of tens of seconds (Section 5.2.3 for further details). On the contrary, custom FPGA-based hardware acceleration of these algorithms can lead to very high performances.

The data stream from the *Stereo camera* is managed by the *Input Controller* (Figure 5.27). The *Input Controller* manages the communication between the *Stereo camera* and the *FPGA subsystem* depending on the protocol supported by the camera. Moreover, it provides two output data streams, each one associated with one of the two acquired images. Each pixel stream is organized in 8-bit packets. Since the camera has a resolution of 8 bpp, every output packet contains one pixel. The output packets associated with the right and left image are provided in input to the two *Noise filter* modules.

The two *Noise filter* instances apply Gaussian noise filtering [82] on the two received images. Image noise filtering is essential to reduce the level of noise in the input images, improving the accuracy of the subsequent feature extraction and matching algorithms. In our architecture, Gaussian filtering is performed via a two-dimensional convolution of the input image with a 7x7 pixels Gaussian kernel mask [82]. For this reasons, this components has been implemented using the *2D-convolver* presented in Section 4.3.1.1. Among the IP-cores contained in the *Image filters* family (Section 4.3.1), it has been decided to use this component since it allows to maintain limited the resources usage allowing to integrate in a single device the overall FPGA sub-system.
In order to perform the desired filtering operation, the *2D-convolver* has been configured according to the input image resolution and the desired kernel size.

The filtered pixels provided by the *2D-convolver*, are sent both to the *Image Enhancer* and to

the external *Shared Memory*. Storing filtered pixels in the external memory is mandatory since this information is needed during the following features matching phase.

Since the illumination conditions in the space environment cannot be predicted a priori and, at the same time, the proposed architecture must always be able to properly work, an image enhancement is required. The enhancement process aims at increasing the quality of the input images in terms of illumination and contrast. In [177] it has been demonstrated that this operation allows to increase the features extraction capability, also in bad illumination conditions.

This task can be performed exploiting one of the IP-cores contained in the *Image Enhancers* family (Section 4.3.3). Since the image conditions cannot be predicted a priori, no assumption can be done on the tonal distribution in the acquired image (i.e., cannot be defined the type of histogram associated with the input image). Thus, in order to design a system able to work autonomously, the *Image Enhancer* module has been implemented using SAFE. As described in Section 4.3.3.3, SAFE is a high performance FPGA-based IP-core that is able to enhance an input image autonomously selecting the best image enhancement technique (i.e., histogram equalization, histogram stretching, or no enhancement) to be applied.

This IP-core receives the input pixels from the *Noise Filter* (Figure 5.27), that provides pixels represented on 8 bit. Since SAFE expects to receive 4 pixels in parallel, an input controller has been added to group the pixels provide by the *Noise Filter* in 4-pixels packets. In addition, SAFE receives in input the two thresholds useful to discriminate on the image enhancement technique to be applied (i.e., *HW* and *BW* described in Section 4.3.3.3).

After image enhancement, the *Rectifier* modules perform the rectification of the left and right image, respectively. Image rectification is essential (i) to remove the image distortion induced by the camera lens (especially in the borders of the image), and (ii) to align the images acquired by two different points of view in order to subsequently apply the epipolar geometry during the feature-matching task [212]. Since the rectification parameters are fixed by the camera type and by the relative orientation between the two cameras, the rectification process can be performed using a simple Look-Up Table (LUT) plus a bilinear interpolation approach, as done in [212] [168], where the two LUTs (one for each image) are stored in the external *Shared Memory*. Basically, each input pixel in position (x, y) in the original image is moved to a new position (x_1, y_1) in the new rectified image. The value of the rectified pixel is computed interpolating the four adjacent pixels in the original image. Since, for resources efficiency reasons, the LUT is stored in the external memory, the coordinates must be translated in an absolute address value. This task is accomplished by simply adding a constant offset, that is equal to the memory base address at which the LUTs are stored.

After image pre-processing, the feature-based stereo-vision algorithm can be applied in order to obtain the features depth map. The algorithm performs feature extraction, feature matching, and finally, feature depth computation. Among these three activities, feature extraction is the most complex one.

As described in Section 5.1, among the available feature extraction algorithms, Harris is probably the best trade-off between precision and complexity [211]. In this specific case, since the two images acquired by the two cameras can present only very small differences in terms of rotations, and almost no differences in terms of scale, its accuracy is comparable to the one provided by SURF and SIFT (Section 2.5.1.1). For this reason, the hardware implementation of the feature extractor and matcher follows the architecture of the namesake components in FEMIP (Section 5.1.1). The SA-FEMIP (Section 5.1.2) architecture has been discarded since well-distributed features is not a key factor in the 3D-shape reconstruction. Moreover, the FEMIP architecture requires less hardware resources than SA-FEMIP.

Finally, the *Feature Depth Calculator* reads the coordinates of the matched features and computes their depth exploiting triangulation [14]. Since matched features are aligned in the two images, the triangulation becomes a two-dimensional problem (i.e., this benefit is obtained from the input image rectification (Section 2.6.1.2)). Looking at Figure 2.28, knowing the focal length of the two cameras, the depth D_i of a feature point P_i, i.e., the distance between the point and the baseline b of the stereo camera, can be computed as:

$$D_i = \frac{f \cdot b}{x_{1,i} - x_{2,i}}$$

where $x_{1,i}$ and $x_{2,i}$ represent the x-coordinate of the considered matched feature P_i in the two acquired images.

Finally, the depth results are stored in the external *Shared Memory* in order to be accessible by the *Processor subsystem* for following computations.

5.2.2.2 Processor sub-system

The *Processor sub-system* includes a processor that executes the novel fast Shape-from-Shading algorithm presented in the previous section. Even if implementing the proposed algorithm directly in hardware, as the modules described in the previous subsection, could lead to a boost of the performances, a software implementation has been preferred to highlight the differences, in terms of execution time, with respect to other state-of-the-art SfS algorithms.

To perform the proposed approach, the processor reads from the shared memory one of the two rectified images. The results of the algorithm represent the shape of the observed object, with respect to a reference plane.

Eventually, the same *Processor sub-system* can be exploited to execute the algorithm that aims at merging the depth information gathered from the *FPGA sub-system*, with the results obtained by the fast SfS approach. This can allow (i) to correct the reconstructed shape in the features points neighbourhoods, and (ii) to extract the absolute size and distance of the object under evaluation.

5.2.3 Experimental Results

To prove the feasibility of the proposed architecture, we implemented both the *FPGA subsystem* and the *Processor subsystem* on a single FPGA device exploring the Aeroflex *GR-CPCI-XC4V* development board, which is equipped with a *Xilinx Virtex 4 XC4VLX100* [226] and a 256 MB SDRAM memory [78]. The choice of using a *Xilinx Virtex 4* FPGA, instead of a more advanced device, fits the considered space debris removal applications. In fact, modern radiation-hardened space-qualified FPGAs exploit the same device architecture [225]. Moreover, the *Processor subsystem* has been implemented using the Aeroflex Gaisler *LEON3* soft-core processor, that represents the standard processor architecture used in space applications [75].

In the following, experimental results are separately reported for the *FPGA subsystem* and the *Processor subsystem*.

5.2.3.1 FPGA subsystem results

All modules described in the previous sections have been synthesized and implemented on the chosen FPGA device, resorting to Xilinx *ISE Design Suite 14.6* [229]. The following table reports the resources consumption of each module composing the *FPGA subsystem*, in terms of Look-Up Tables (LUTs), registers (FFs), and internal Block-RAMs (BRAMs) (Section 3.3).

Table 5.5: FPGA subsystem modules logic and memory hardware resources consumption for a *Xilinx Virtex 4 XC4VLX100* FPGA device.

	LUTs	FFs	BRAM
Input Controller	352	96	-
Noise Filter (x2)	11,792	1,392	14
Image Enhancer(x2)	2,635	317	16
Rectifier (x2)	1,216	712	8
Features Extractor (x2)	19,162	2,212	12
Features Matcher	2,432	656	19
Features Depth Calculator	615	64	-
Total	**38,204** (38.9%)	**5,449** (5.54%)	**69** (28.8%)

The numbers in brackets represent the percentages of resources used with respect to the total available in the FPGA device. It is worth noting that *Noise Filters*, *Image Enhancers*, *Rectifiers*, and *Features Extractors* are instantiated twice in the design.

To emulate the camera, the images are supposed to be pre-loaded in an external memory. Thus, the *Input Controller* consists of a Direct Memory Access interface that autonomously reads the images pre-loaded into the *Shared Memory*.

The entire processing chain is able to process a couple of images and to provide the associated depth map in about $32ms$, leading to a throughput of 31 image couples per second.

The bottleneck of the system is represented by the external *Shared Memory*, that in some time slots is simultaneously requested by different modules. Using a dual-port memory can help to avoid the stall of the processing chain, leading to a greater throughput.

To highlight the speed-up obtained by resorting to hardware acceleration, the algorithm implemented by the proposed hardware modules have been also described in C, and compiled (using the maximum possible optimization level) for the *LEON3* processor. The processor has been implemented enabling the floating-point unit and the internal data/instruction cache memories of 4 KB and 16 KB, respectively. The overall software execution time attests around 42 seconds, when the processor runs at 60 MHz (i.e., the maximum operating frequency of the LEON3 processor implemented on the selected FPGA device). The major contribution in the execution time is given by the Gaussian filtering and feature extraction functions, that perform two-dimensional convolution.

Comparing the overall software and hardware execution times, hardware acceleration provides a speed-up of 1,300x.

Finally, focusing on the *Feature Extractor* and *Feature Matcher* modules, we can highlight the gain, in terms of hardware resources consumption, of using the *Harris* algorithm, with respect to more complex *SIFT* or *SURF* extractors. As an example, [16] and [25] propose two FPGA-based implementations of the SURF algorithm. The architecture proposed in [16] consumes almost 100% of the LUTs available on a medium sized *Xilinx Virtex 6* FPGA, without guaranteeing real-time performances. Similarly, the architecture proposed in [25] consumes about 90% of the internal memory of a *Xilinx Virtex 5* FPGA. It saves logic resources, but it is able to process in real-time only images with a limited resolution of 640 x 480 pixels. Another example is presented in [233] where an FPGA-based implementation of the SIFT algorithm is presented. It is able to process in real-time 640 x 480 pixel images, consuming about 30,000 LUTs and 97 internal Digital Signal Processor hard macros in a *Xilinx Virtex 5* FPGA. Instead, taking into account the reduced complexity of the *Harris* algorithm, the feature extraction for the two 1024x1024 input images, and the matching task can be performed in real-time using only 21,594 LUTs and 31 BRAMs resources, representing the 22% and the 12.9% of the considered *Xilinx Virtex 4* FPGA device, respectively.

An explicit comparison with other state-of-the-art FPGA-based stereo-vision architecture has not been made since they focus on block-based methods [38, 206, 217, 243]. Even if they provide more dense results, due to their increased complexity they incur in a greater hardware resources consumption (not including the resources needed for the image pre-processing) [25][233][38].

5.2.3.2 Processor subsystem results

The *Processor subsystem* has been implemented resorting to the *LEON3* soft-core processor architecture [75], and integrated in the same FPGA device as the *FPGA subsystem*. The processor has been implemented enabling the floating-point unit and the internal data and instruction cache

memories of 4 KB and 16 KB, respectively. The maximum operating frequency of the processor, synthesized on a *Xilinx Virtex-4 XC4VLX100* [226], is equal to 60 MHz.

The chosen processor configuration leads to an hardware resources usage of 21,395 LUTs, 8,750 FFs, and 32 BRAMs. Thus, the overall resources consumption of the proposed system (FPGA and Processor subsystems) is around 60% of the overall logic resources available in the FPGA device.

In order to evaluate the execution time of the proposed *Fast-SfS* algorithm, different executions have been performed on the *LEON3* processor, providing in input images with different size. The graph in Figure 5.28 shows the execution time trend with respect to the input image size.

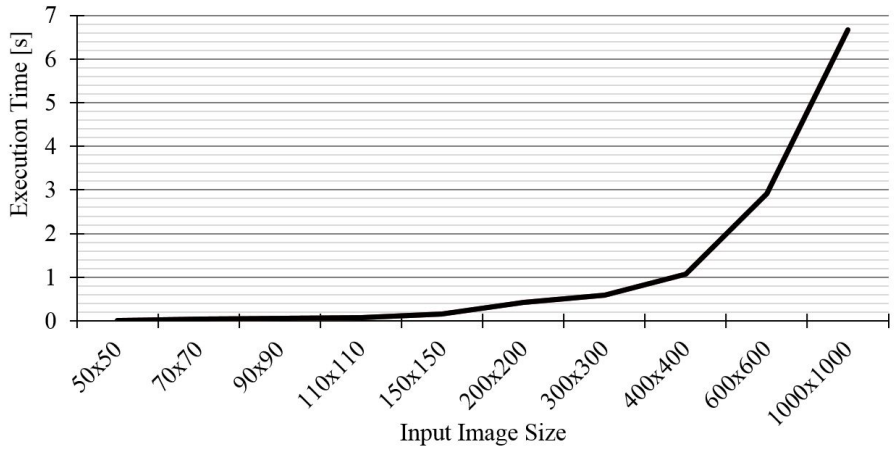


Figure 5.28: Proposed algorithm execution time trend with respect to the input image size.

The proposed algorithm has been compared, in terms of execution time, to other Shape-from-Shading approaches proposed in literature. [60] reports execution times of the fastest SfS algorithms presented in literature. The algorithms are executed on a Sun Enterprise E420 machine [96], equipped with four UltraSparc II processors running at 450 MHz, each one with 4 MB of dedicated cache. From [60], the fastest SfS algorithm is the one presented in [171], that is an iterative algorithm, requiring 0.29 s for 5 iterations and 1.17 for 20 iterations (to ensure better results), to process a 256x256 images. Even if running on a processor clocked at 1/6 of the operating frequency, and equipped with a cache of 3 orders of magnitude smaller, the proposed algorithm requires almost the same time of [171].

In [194] authors state that the proposed non-iterative SfS algorithm is faster than the iterative ones. By comparing their approach with *Fast-SfS*, from Figure 5.29 it can be seen that the speed-up is always greater than 3x. It is worth noting that in [194] the testbed used to run the proposed algorithm is not defined.

The algorithm applied on the two synthetic images provides good results since they are char-

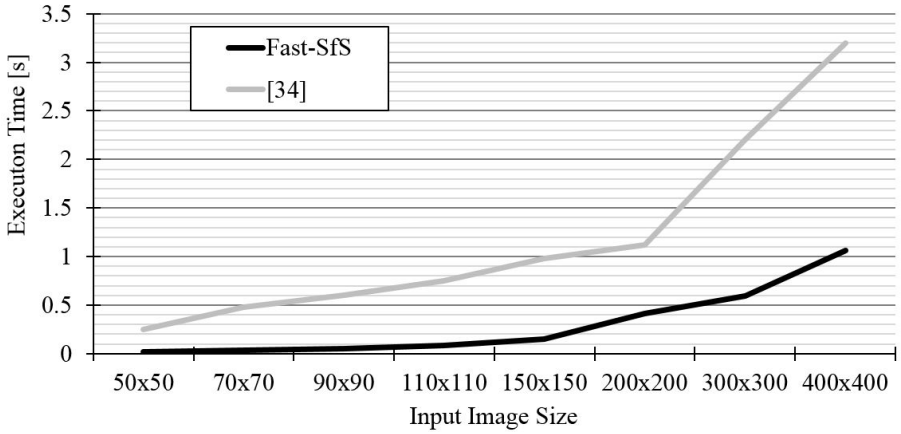


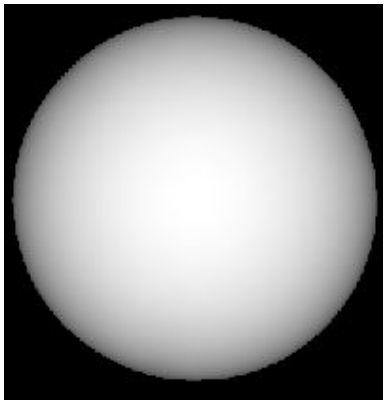
Figure 5.29: Comparision of the execution times of *Fast-SfS* and the algorithm reported in [194].

acterized by a diffused light and a monochromatic surface. This characteristics completely match the assumptions on which the proposed algorithm is based (Section 5.2.1).

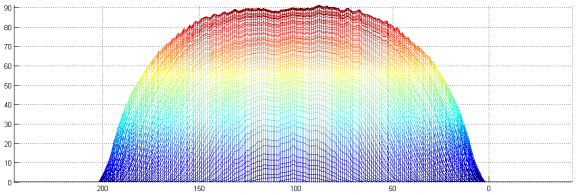
On the other hand, in the real image the target object is illuminated by direct light, since diffused light cannot be easily reproduced in laboratory. Moreover, due to the reflective material composing the object, the image presents some light spots (see the upper central part of the drinking bottle in Figure 5.32(a)). As can be noted in Figure 5.32(b) and 5.32(c), these light spots lead to some peaks in the reconstructed shape. Nevertheless, the resulting shape provided by the Fast-SfS algorithm follows the profile of the real object and the peaks can be efficiently corrected by merging the reconstructed shape with the information gathered from the stereo-vision algorithm.

5.3 FPGA-based JPEG-LS Image Compressor

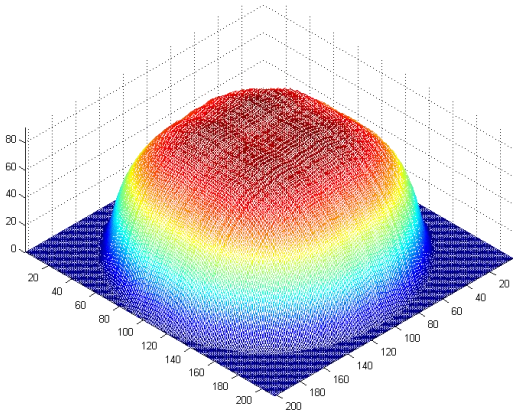
This section presents the main outcome of my visiting period at ESTEC: an area-efficient FPGA-based implementation of the JPEG-LS algorithm (Section 2.7.1). The main motivation above this work concerns the increasing need to compress images taken during space exploration mission. In fact, in the last years, all space agencies increased the number of cameras mounted on spacecraft and rover involved in space exploration mission to take a plenty of images. In the past, images acquired during space mission have been just exploited for scientific purposes, instead in the last period, space agencies started to use these images for public relation purpose, also, in order to increase the social interest in the space agency activities. This trend directly reflected in an increased amount of picture taken in space environment. However, the strict constraint in terms of mass and power consumption did not allow to increase in the same way the memory resources and communication bandwidth to send picture to Earth. For the aforementioned rea-



(a) Input image



(b) *Fast-SfS* results (side-view)

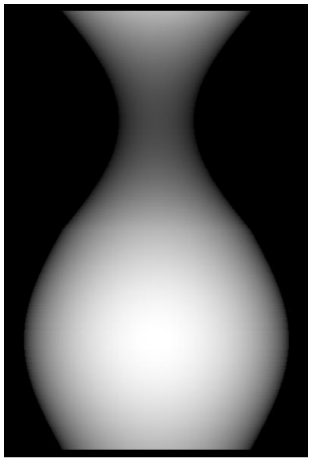


(c) *Fast-SfS* results (dimetric-view)

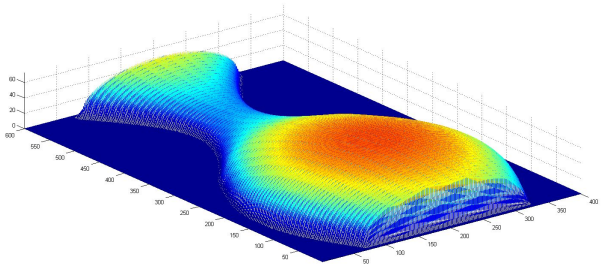
Figure 5.30: *Fast-SfS* output result on the semi-sphere image

sons, the need of efficient systems able to reduce the size of images while preserving the image quality increased more and more.

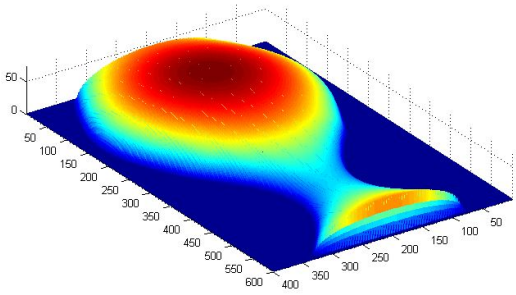
Following this growing need, ESTEC people decided to develop an area-efficient hardware system able to strongly reduce the image size while introducing a negligible loss of information. In addition the required system must be able to tune the applied compression factor depending on



(a) Input image



(b) *Fast-SfS* results (dimetric-view)

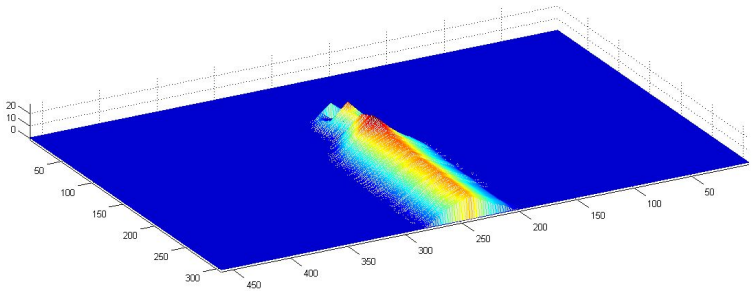


(c) *Fast-SfS* results (opposite dimetric-view)

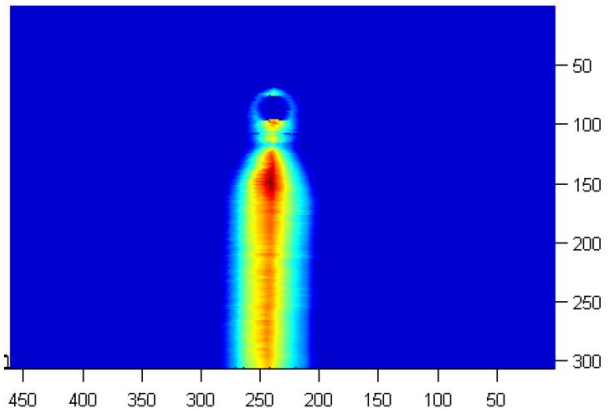
Figure 5.31: *Fast-SfS* output result on the vase image

the actual availability in terms of memory resources and communication bandwidth to Earth. According to these requirements and the characteristics of the image compression algorithms presented in Section 2.7, the target algorithm for this image compression system has been the JPEG-LS Algorithm.

(a) Input image



(b) *Fast-SfS* results (front-view, highlighting the light spots)



(c) *Fast-SfS* results (oblique-view)

Figure 5.32: *Fast-SfS* output result on the drinking bottle image

Following sections present the internal block diagram of the proposed system and the achieved performances in terms of hardware resources usage, compression capability and introduced loss of information on the compressed images.

5.3.1 JPEG-LS hardware architecture

This section describes the FPGA-based IP-core implementing the near-lossless JPEG-LS algorithm presented in Section 2.7.1. It has been decided to implement only the near-lossless mode, since, as demonstrated in Section 5.3.2, with $NEAR = 1$ (i.e., the lowest amount of information loss) the quality of the compressed image is practically the same as the original one. Moreover, this decision ensures to reduce the hardware complexity, and increase the achievable output throughput. Moreover, to increase the flexibility and the applicability, the proposed hardware system can be customized, exploiting VHDL generics [232], to support the compression of color and gray-scale images characterized by different image resolutions (i.e., image size, and bit-per-pixel resolution).

Figure 5.33 shows the main modules composing the implemented hardware component.

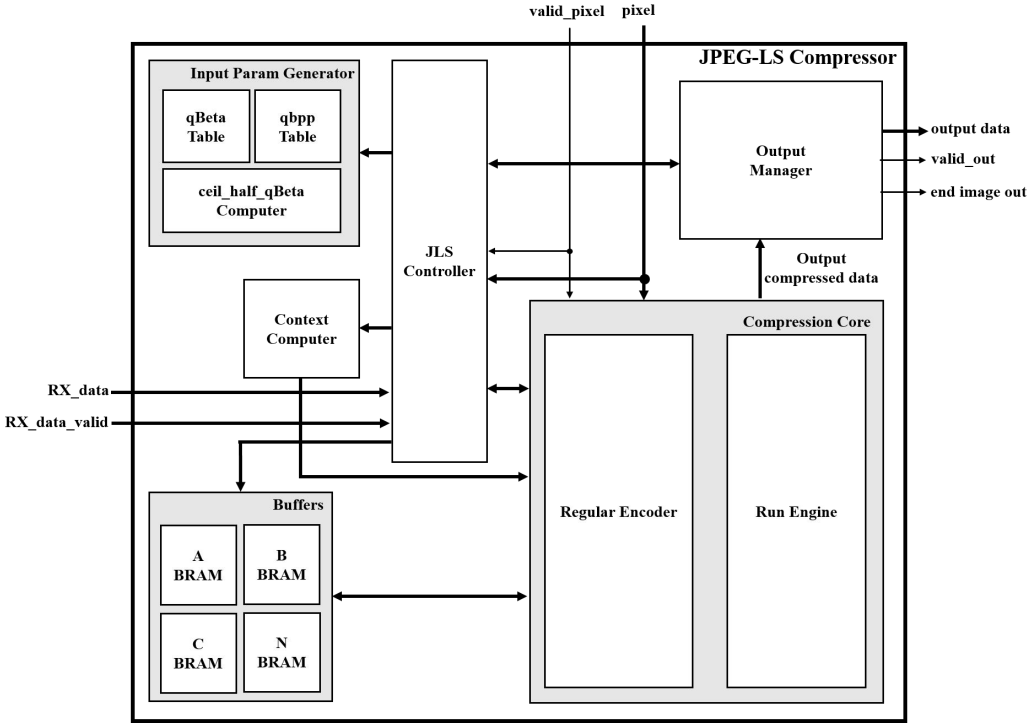


Figure 5.33: JPEG-LS IP-core architecture

The IP-core receives in input four signals:

- *pixel*: the pixel value to be processed. The input image is supposed to be received in input in a raster way (i.e., pixels are provided from top to bottom, and from left to right), as provided by almost all camera.
- *valid_pixel*: a one bit signal asserted when the data on *pixel* is valid.

- *RX_data*: an 8 bit command signal aiming at providing in input to the compressor: (i) the desired $NEAR$ parameter (Section 2.7.1), (ii) the start compression command, and (iii) the acknowledge that all the compressed bytes have been received correctly.

- *valid_RX_data*: a one bit signal asserted when the input command signal is valid

In the current implementation the accepted $NEAR$ values are from 1 to 5. As demonstrated in Section 5.3.2, this range ensures to reach a high compression ratio, and, at the same time, maintain the image quality really high.

The output compressed image is provided by means of three signals: (i) *output data* that is an 8 bit signal containing the byte composing the output compressed bitstream, (ii) *valid_out* is a one bit signal asserted when the*output data* signal is valid, and (iii) *end_of_image* that is asserted when the last valid byte has been provided in output.

Internally, the proposed architecture is composed of six main components:

- *JLS Controller* is the manager of all the operations inside the *JPEG LS compressor*. It manages the inputs and it synchronizes the computation by providing the data to the different internal modules in the proper timing sequence.

- *Input Param Generator* aims at computing all the input parameters depending on the received $NEAR$ value. Basically, this module computes the $qbpp$, $qBeta$, and the $ceil\_half\_qBeta$, as described in Section A.0.2.1. For this reason this module operates just before the start of the image compression task when a new $NEAR$ value is received.

- *Context Computer* computes the context value for the *regular encoding* mode. This module receives in input, from the *JLS Compressor*, the reconstructed sample values associated with the next pixel to be processed, and it produces the $ClassMapcontext$ value as described in Section A.0.2.2.

- *Buffers* aims at storing the values composing the A, B, C, and N vectors (Section A.0.2.1). Thus, this component is just a set of BRAMs that ensure to internally store these vectors without requiring accesses to the external memory.

- *Compression Core* is the module that actually performs the compression of the input pixels. As shown in Figure 5.33, it is internally composed of two main modules: *Regular Encoder* and *Run Engine*, that respectively perform the regular (Section A.0.2.3) and run-length encoding (Section A.0.2.4) compression. These two components operate in parallel for every received pixel producing a pixel compressed value for both compression method. However, thanks to the information provided by the *JLS Controller*, *Compression Core* provides in output just the data associated with the pixel compressed with the selected compression mode (Section A.0.2.2).

- *Output Manager* receives the output of the compression process, generated by the *Regular Encoder*, and the *Run Engine*, and it provides the output compressed bytes according to the format imposed by the standard [101].

As can be noted, differently from the description of the other proposed hardware systems, the level of provided internal architecture details is lower. This is mainly due to an *Non Disclosure Agreement* (NDA), signed with the ESTEC security office, that protect the architecture of this hardware system and it does not allow to provide more detailed information about the hardware module implementation.

5.3.2 Experimental results

To evaluate the hardware resources usage and the timing performances, the proposed architecture has been synthesized resorting to *Xilinx ISE Design Suite 14.6* [229] on a *Xilinx Virtex 4 XC4VLX100 FPGA* [226]. Post place and route simulations have been done with Modelsim SE 10.1c [83].

Table 5.6 compares the area occupation of proposed architecture w.r.t. the actual state-of-the-art implementations [113] [218].

Table 5.6: Resource Usage for *Xilinx Virtex 4 XC4VLX100 FPGA device*

IP-core	**Resource Usage**	
	Slices	**LUTs**
Proposed	1,926 (3.21%)	3,095 (3.14%)
[113]	2,154 (4.38%)	5,402 (5.50%)
[218]	2,596 (5.28%)	6,511 (6.62%)

As can be noted, the proposed implementation outperforms in terms of resources usage the current state-of-the-art implementations. In particular, the improvements is around 11% for the used Slices, and 74% for the used LUTs. This gain is even bigger if we consider that the implementations presented in [113] and [218] are only able to perform the Lossless JPEG-LS compression, that requires a lower number of operations w.r.t. the Near lossless one. Instead, the proposed implementation is able to perform a Near-lossless compression in which the compression level can be externally programmed.

Concerning the timing performances in the proposed implementation the logic performing the actual compression is able to reach a clock frequency of 30 MHz. Since the output throughput is of 1 compressed pixel every 3 clock cycle, the proposed IP-core is able to reach in the worst case (i.e., when all the pixel are compressed with the regular encoding) a throughput of 10 *fps* processing an image with a resolution up to 1,024 x 1,024 pixel. The achieved throughput is particularly

important since is the maximum frame rate to apply an Intra-frame compression algorithm (Section 2.7).

In order to quantify the accuracy of the *JPEG-LS compressor*, in terms of compression capability and loss of information introduced on the compressed image, a dataset of images have been provided in input to the proposed hardware system.

The selected datasets of images includes a comprehensive selection of images taken in Mars environment. The Mars environment has been selected since, as aforementioned, the peculiarity of the JPEG-LS algorithm are of particular importance especially for space exploration missions, like Mars missions. In order to model the images that can be taken during a Mars exploration mission, the input image dataset has been composed with images representing the Mars surface (this kind of images model the picture that can be taken during the descending phase on Mars), and images taken on the Mars surface.
The first set has been filled with images extracted from the *Mars Express* database [69], while the second with images extracted from the *NASA's Mars Rover Curiosity on Red Planet* database [133]. In the sequel we refer to these two set of images as *Mars Express* and *Curiosity* datasets, respectively.
Figures 5.34 and 5.35 show an example of the images contained in the two datasets.

Figure 5.34: *Mars Express* dataset examples

As can be noted from Figures 5.34 and 5.35, to properly test the proposed system under different conditions, in both cases the images are characterized by different tonal distribution.
Moreover, for the set of images taken on the Mars surface, the selected pictures contains different

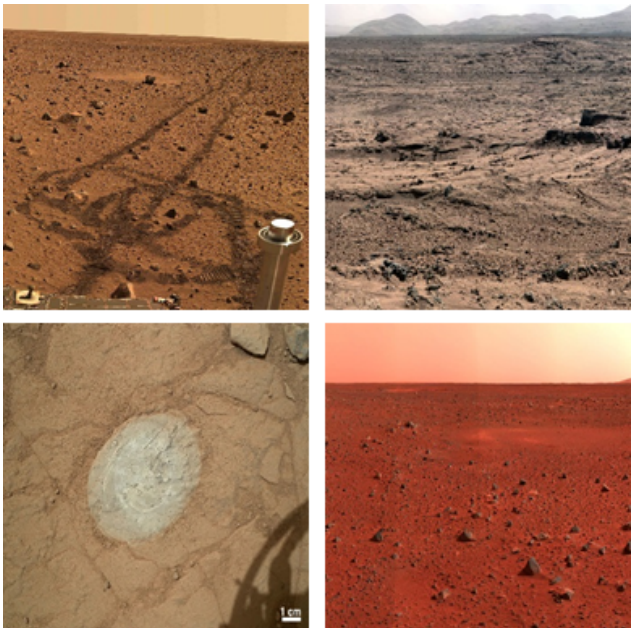
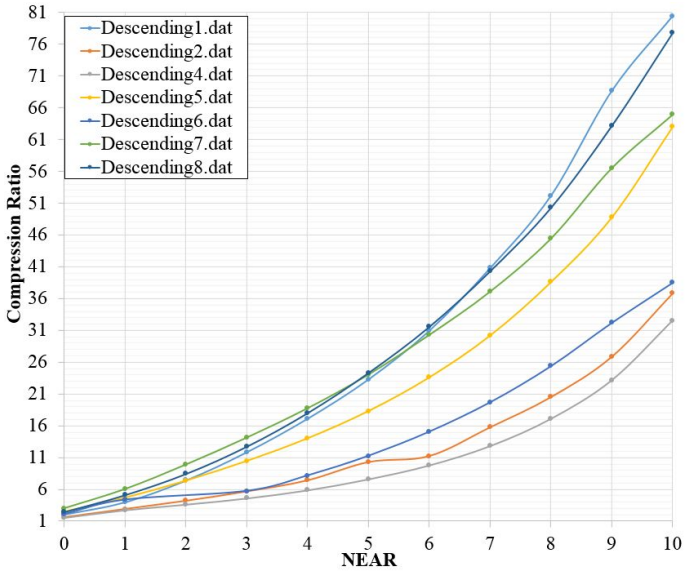


Figure 5.35: *Curiosity* dataset examples

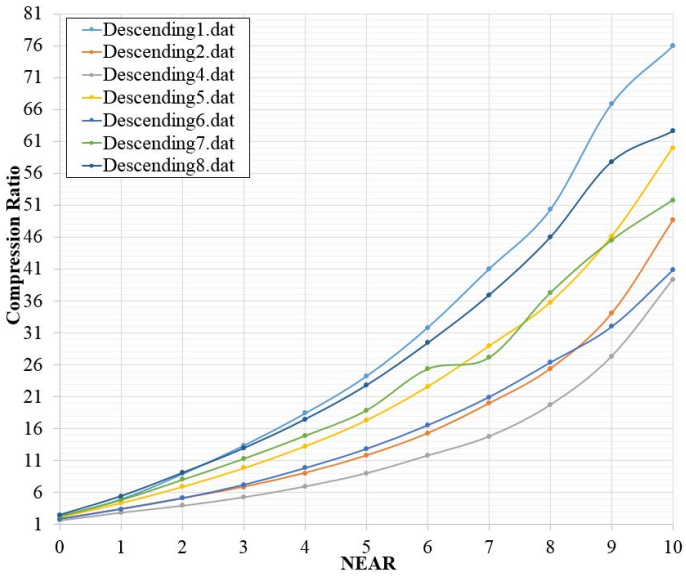
portion of sky, this allows to force the *JPEG-LS compressor* to use more or less the run encoding mode (the sky represent a flat zone of the image, so in the sky portion of the image the run encoding is extensively used (Section A.0.2.4)). In other words, the system can be tested under different working conditions.
Moreover, for better characterize the system performances, the pictures composing the two datasets have been manipulated. In particular, the images have been resized to the 1,024 x 1,024 and 640 x 480 image resolutions, and, for each image resolution, the bit per channel resolution has been set to 8 bpc, and 5 bpc (i.e., the most common parameters for space cameras). For these reasons, 4 new sets of images have been created for both the *Mars Express* and *Curiosity* dataset.

All of these dataset has been provided in input to the proposed *JPEG-LS Image Compressor* to quantify the achieved compression rate.
In Figure 5.36 and 5.37, the graphs showing the trends of the compression rate vs. the $NEAR$ parameter for each image dataset are reported.

As can be noted, the compression rate is not so much influenced by the image resolution, in fact comparing Figure 5.36(a) with Figure 5.36(b), and Figure 5.37(a) with Figure 5.37(b), the compression rate maintains almost the same trend in both the 1,024 x 1,024 and 640 x 480 image resolutions.
Instead, comparing the compression rate inside the same graph, it can be noted that the difference between two images can be very high. This effect is due to the number of flat zones that can be found inside an image. In fact, higher is the number and the size of the flat zones inside a



(a) 1,024 x 1,024

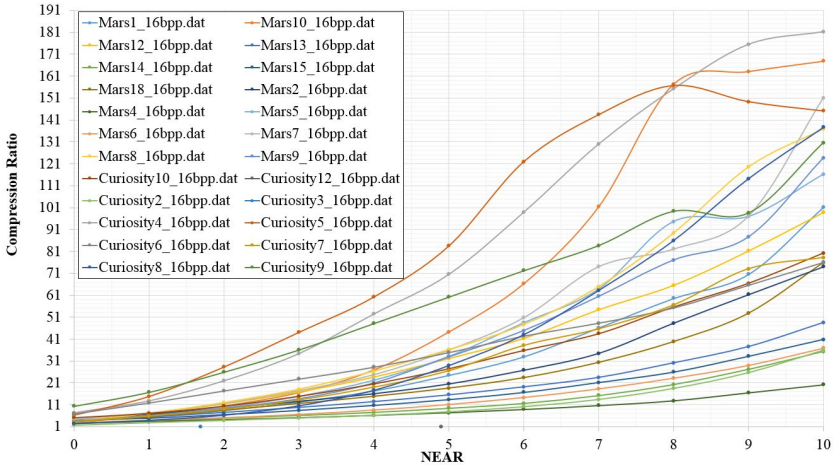


(b) 640 x 480

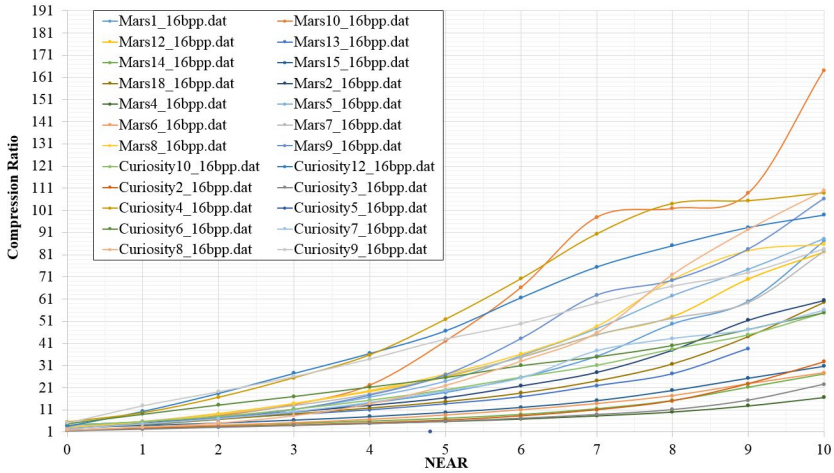
Figure 5.36: 5 *bpc Mars Express* dataset compression rate

picture, higher will be the length of each run in the run encoding mode (Section A.0.2.4), and so smaller will be the size of the compression image.

Somehow, as shown in Table 5.7, analysing the worst case for each image dataset the compression rate reach always good performance, demonstrating the high compression capability of the



(a) 1,024 x 1,024



(b) 640 x 480

Figure 5.37: 5 *bpc Curiosity* compression rate

JPEG-LS algorithm.

To evaluate the error introduced on the compressed image in the case of near-lossless compression, the *Peak Signal to Noise Ratio* (PSNR) has been compute for every compressed image composing the datasets.

PSNR is defined as the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Because many signals have a very wide dynamic range, PSNR is usually expressed in terms of the logarithmic decibel scale. Equation 5.3 shows the formula to compute this parameter.

$$PSNR = 20\log_{10}\frac{MAX_I}{MSE} \tag{5.3}$$

Table 5.7: Worst case compression rate

$NEAR$	***Figure 5.36(a)***	***Figure 5.36(b)***	***Figure 5.37(a)***	***Figure 5.37(b)***
0	1.56	1.58	2.58	1.62
1	2.76	2.82	3.49	2.73
2	3.64	3.90	4.29	3.51
3	4.61	5.22	5.17	4.19
4	5.88	6.91	6.20	4.94
5	7.60	8.98	7.44	5.84
6	9.79	11.76	8.90	6.91
7	12.82	14.75	10.68	8.35
8	17.11	19.69	12.85	10.11
9	23.16	27.28	16.50	12.88
10	32.52	39.34	20.16	16.67

where MAX_I is the maximum possible pixel value of the image (i.e., $2^{bpp}-1$), and MSE is the *Mean squared error* that can be computed as in Equation 5.4.

$$MSE = \frac{1}{m*n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(I(i,j)-DI(i,j))^2 \tag{5.4}$$

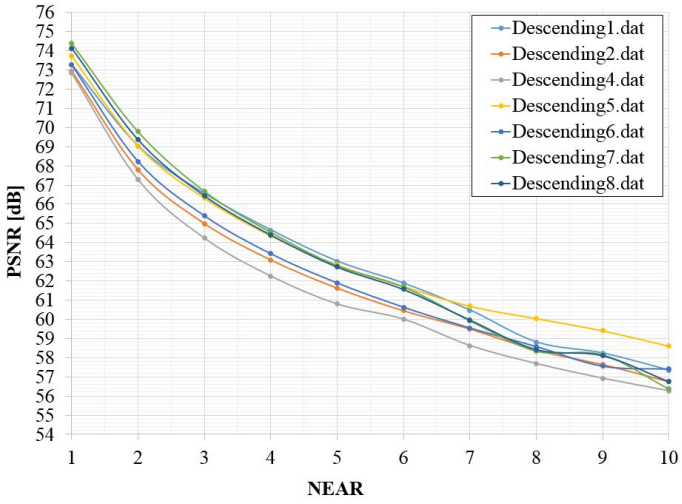
where m and n are the number of row and column composing the image, respectively, $I(i,j)$ is the pixel value of the original image in the (i,j) position, and $DI(i,j)$ is the pixel value in the decompressed image in the (i,j) position.

Figure 5.38 and 5.39 show the computed PSNR parameters on the two datasets. As can be noted, from both figures, the PSNR parameter, as the compression rate, is not so much influenced by the image resolution. Moreover, unexpectedly, this parameter is less influenced by the different kinds of images than the compression rate parameter.

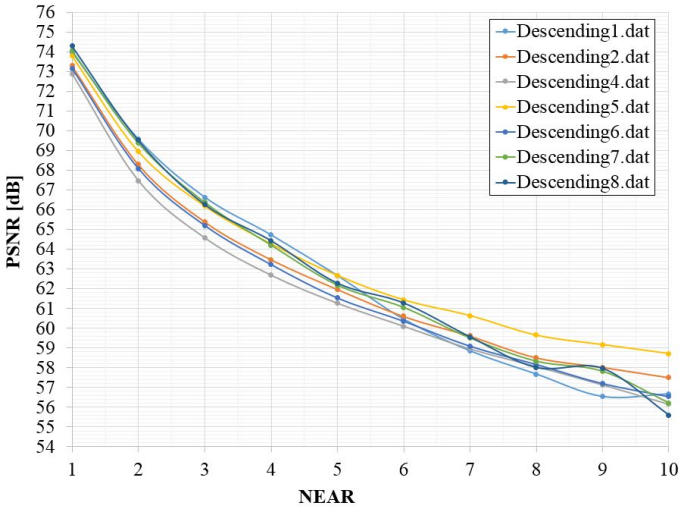
In general, the data gathered about the PSNR parameter demonstrate the effectiveness of the near-lossless compression, that allows to increase the achieved compression rate while maintaining limited the introduced error on the compressed image.

Analysing the literature, the common PSNR characterizing a lossy compression algorithm can vary from 50 to 30 dB [184] [27]. Comparing this value with the data contained in the graph in Figure 5.38 and 5.39, the near-lossless compression is able to introduce a lower error on the compressed image, in fact, even in the worst case, the PSNR varies from 71 to 53 dB.

For the sake of completeness, in the sequel some examples on how the compressed image look like are reported.

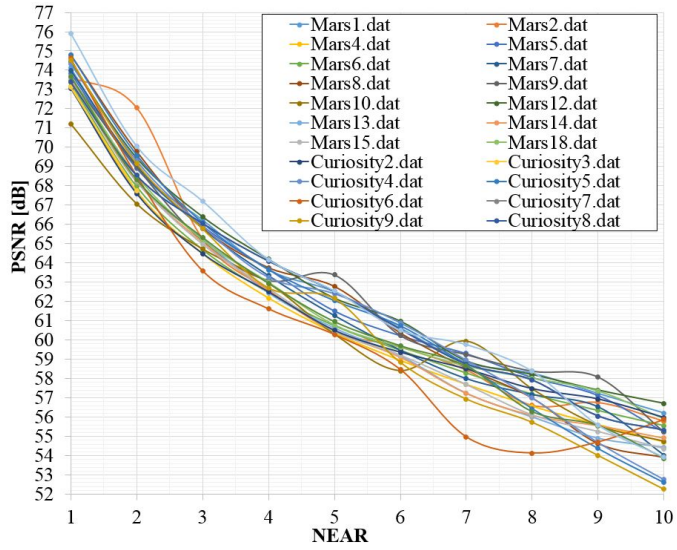


(a) 1,024 x 1,024

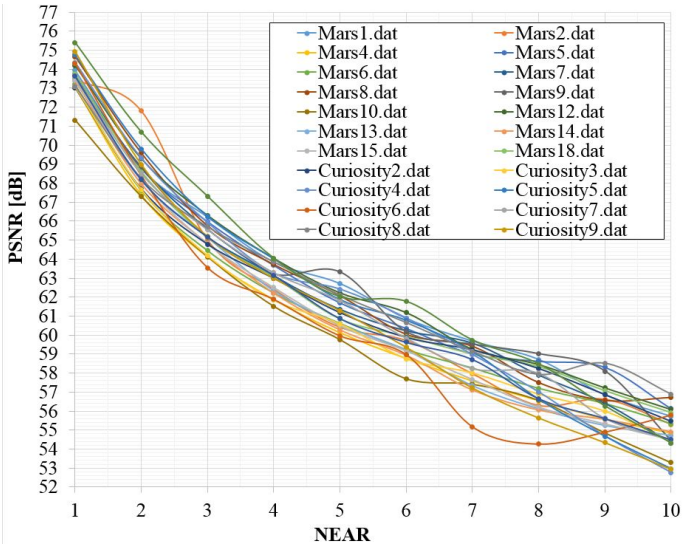


(b) 640 x 480

Figure 5.38: 5 *bpc Mars Express* dataset PSNR parameter

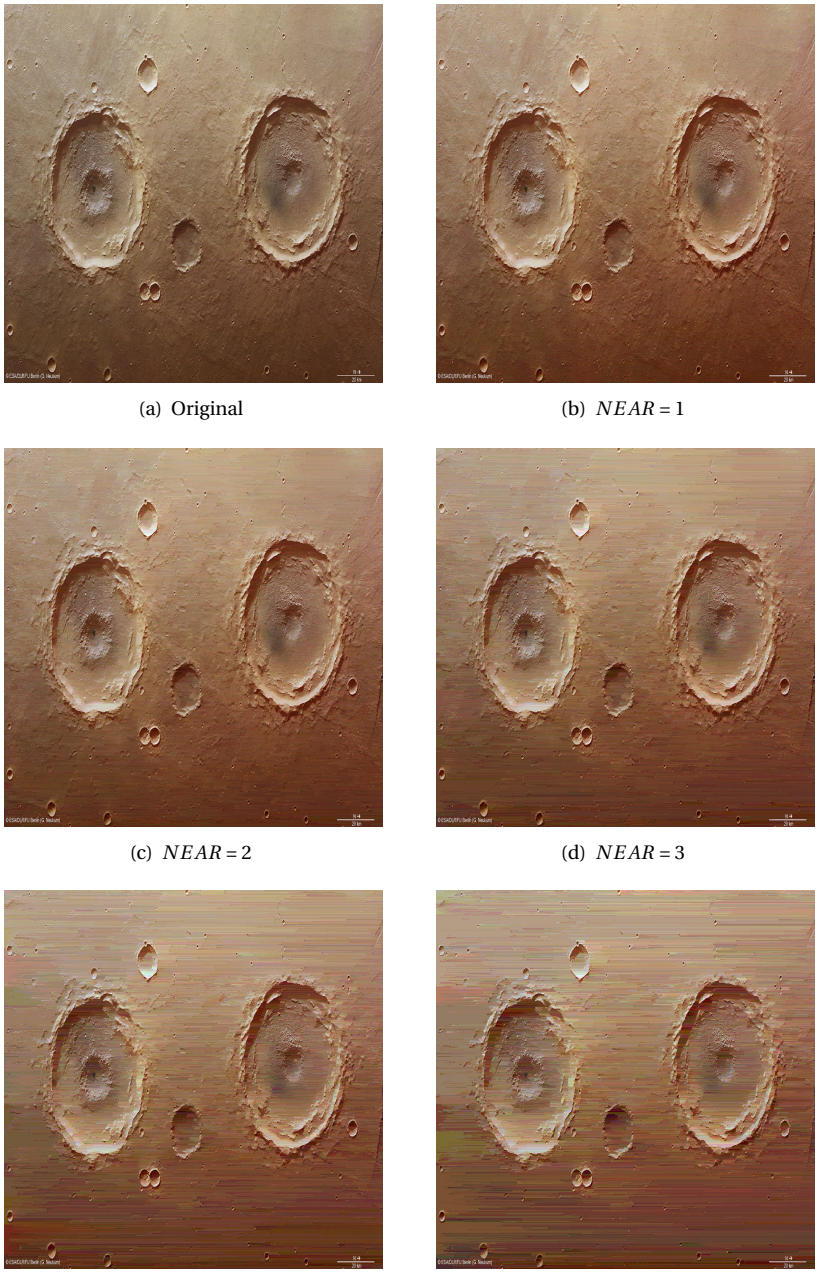


(a) 1,024 x 1,024



(b) 640 x 480

Figure 5.39: 5 *bpc Curiosity* PSNR parameter

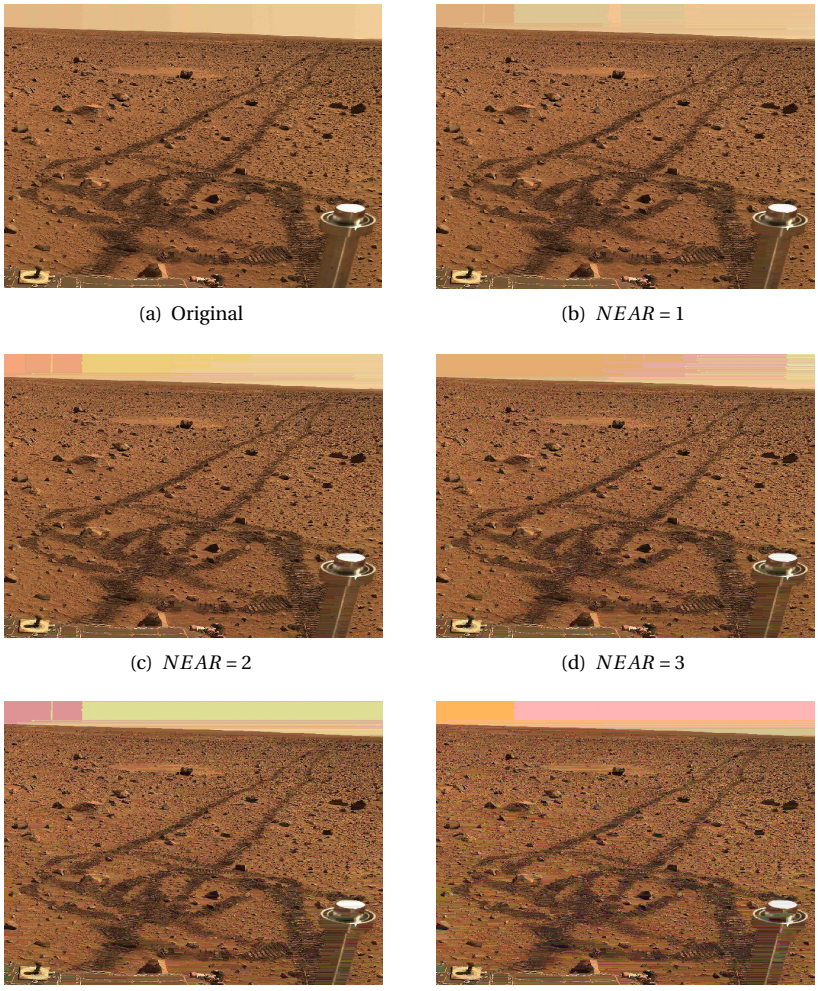


(a) Original (b) $NEAR = 1$

(c) $NEAR = 2$ (d) $NEAR = 3$

(e) $NEAR = 4$ (f) $NEAR = 5$

Figure 5.40: *Mars Express* database compression example



(a) Original (b) $NEAR = 1$

(c) $NEAR = 2$ (d) $NEAR = 3$

(e) $NEAR = 4$ (f) $NEAR = 5$

Figure 5.41: *Curiosity* database compression example

CONCLUSIONS

In this thesis have been presented a library of FPGA-based IP-cores that can ease the design of complex image processing systems to be used in space applications.

A preliminary introduction on the importance of digital image processing in space applications allows the reader to understand how the proposed work can find a plenty of applications. All the applications in which image processing is a key element have been presented. A major effort have been spent for the applications on which my research activity had a big impact. In fact, for them have been reported a detailed description of the involved algorithms in order to allow the reader to understand the level of complexity of the operations to be executed. This ensures to better realize why to reach the mission requirements, in terms of speed, high performance hardware accelerators are mandatory.

The comparison between FPGAs and ASICs highlights the benefits, in terms of flexibility and cost reduction, provided by the former technology, and so justifies the trend of all space agencies to prefer FPGAs for accelerating the execution of time consuming tasks. A detailed presentation of the available space-grade FPGA technologies provided the needed knowledge to understand why SRAM-based FPGAs are the most promising technology for the image processing acceleration in space. In fact, this technology, even if it is less robust against transient error caused by the harsh space environment, embraces all the needs, in terms of flexibility and high performances, required by future space missions. For the sake of completeness, a comprehensive presentation of the fault-tolerance techniques that must be adopted to use SRAM-based FPGA has been reported.

After this introduction part, the core of my research work is presented: the FPGA-based IP-core library for high performance image processing in space. First, the motivation about the importance of IP-core libraries in the design flow of complex system is presented. Then, it has been described the methodology that allowed to reach the required performances. This methodology guides the designer from the image processing algorithm selection up to the hardware implementation. During the overall design flow, it suggests at which stage different optimizations (i.e., algorithm modification, internal parameters boost, data format and representation, hardware architectural approaches etc.) must be applied. Moreover, in addition to the hardware implementation, the proposed methodology outcomes two models (i.e., algorithm and hardware models) that can be used as inputs to the proposed verification and validation approach to quickly obtain a validated IP-core.

Thanks to this methodology, a library of high performance FPGA-based hardware accelerators have been developed. The proposed FPGA-based IP-core library has been organized in families.

Each family contains a set of IP-cores that strongly accelerate the computation of an elementary image processing task. For each IP-core a detailed presentation of the internal architecture has been reported. This description depicts the architectural approaches exploited to achieve the required performances. Moreover, to better highlight the achieved high performances for each IP-core the area occupation and speed have been reported.

Eventually, to demonstrate the benefits provided by the proposed library in the development of complex image processing system for space, the developed image processing systems have been presented. These systems are the results of strong collaboration between me and space industries. The main purpose of this collaboration is the acceleration of applications that currently require the highest performance in space (i.e., the relative VBN, the Active Space Debris removal, and the image compression). In addition to the proposed IP-cores, each system is composed by custom hardware components and software routines running on the on-board computer. This mix of hardware and software elements demonstrates the flexibility and the integrability of the proposed IP-cores in different systems. In order to demonstrate the impact of my research activity, all the developed systems have been compared with the current state-of-the-art implementations, in terms of both hardware performances and adaptability to the unpredictable environmental and image conditions. These comparisons highlight how my work provided an important boost on the performances of image processing in space applications.

To conclude this thesis, it must be recognize that the image processing in space is increasing more and more its importance in space applications, so to even more increase the relevance of future space missions, and in particular space exploration ones, a lot of research work still to be done. However, this thesis can be considered a solid starting point for next research activities in this field.

APPENDIX A

JPEG-LS ALGORITHM DETAILS

In this appendix a detailed description of the operations performed by the JPEG-LS algorithm is reported. In addition to the main operations mentioned in Section 2.7.1, this appendix, in Section A.0.2.1, provides a deep explanation of all the internal parameters used by the JPEG-LS algorithm.

A.0.2.1 Parameters initialization

The *Parameters initialization* phase aims at assigning the initial value to each JPEG-LS internal parameters and variables. This phase must be repeated before each image compression task.
The main variables and parameters to be initialized are:

- *NEAR*: difference bound for near-lossless coding. Usually this parameter is not initialized, but it is externally provided by the user depending the desired level of compression to be applied on the input image. If $NEAR$ is zero the lossless image compression is performed, otherwise the near-lossless one.

- *qBeta*: an internal constant used during the Error Mapping phase (Algorithm 12). The value of this constant is computed as follow:

$$qBeta = (alpha + 2 * NEAR + QUANT - 1)/QUANT \tag{A.1}$$

  where $alpha$ is the maximum value that can be assumed by an input pixel, and $QUANT$:

$$QUANT = 2 * NEAR + 1 \tag{A.2}$$

- *ceil_half_qBeta*: another internal constant used during the Error Mapping phase (Algorithm 12). The value of this constant is computed as follow:

$$ceil\_half\_qBeta = \frac{qbeta + 1}{2} \tag{A.3}$$

- *qbpp*: an internal constant defining the number of bits needed to represent a mapped error value. Its value is equal to the bpp resolution (i.e., the number of bit used to represent pixels in the input image) in the lossless compression, instead, in the near-lossless one, the $qbpp$ value is computed as follow:

$$for(qbpp = 1; (2^{qbpp}) < qBeta; qbpp++) \tag{A.4}$$

- *LIMIT*: the length parameter used during the Golomb coding phase (Algorithms 13 and 22). Its value is initialized to:

$$LIMIT = \begin{cases} 2*(bpp+8) - qbpp - 1, & \text{if } bpp < 8 \\ 4*bpp - qbpp - 1, & \text{if } bpp \geq 8 \end{cases}$$

- *J[0..31]*: is a 32-cells constant vector used during the block-MELCODE coding (Section A.0.2.6). Its initial value is:

$$J[0..31] = 0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3,4,4,5,5,6,6,7,7,8,9,10,11,12,13,14,15 \tag{A.5}$$

- *A[0..366]*: it is a 367-cells vector used during the encoding phase to save the counters storing accumulated prediction error magnitudes. The cells from 0 to 364 are used in the regular encoding, instead the 365 and 366 ones are associated with the run mode interruption context (Section A.0.2.4). All the cells of this vector are initialized to 2.

- *B[0..366]* it is a 367-cells vector used during the encoding phase to store bias values used to update the prediction correction variable (C[0..364]). The cells from 0 to 364 are used in the regular encoding, instead the 365 and 366 ones are associated with the run mode interruption context (Section A.0.2.4). All the cells of this vector are initialized to 0.

- *C[0..364]*: it is a 365-cells vector used during the encoding phase to store prediction correction values. This variable is used during the regular encoding, only, to perform the correction of the predicted value (Algorithm 7). All the cells of this vector are initialized to 0.

- *N[0..366]*: it is a 367-cells vector used during the encoding phase to store counters for frequency of occurrence of each computed pixel context (i.e., $ClassMapcontext$ in regular encoding, and Q in run encoding). The cells from 0 to 364 are used in the regular encoding, instead the 365 and 366 ones are associated with the run mode interruption context (Section A.0.2.4). All the cells of this vector are initialized to 1.

A.0.2.2 Context Modelling

As shown in Figure 2.30, the first operation to be performed is the *Context Modelling*. This procedure determines a probability distribution used to encode the current pixel (*Ix* in Figure 2.31). In particular, the context is determined from four neighbourhood reconstructed samples (Ra, Rb, Rc, and Rd in Figure 2.31). As can be noted from Figure 2.31, each time a pixel is compressed a new reconstructed sample is associated with it, and this value is used as Ra for the next processed pixel. Obviously, if the current sample is in the first row, in the first column, or in the last column, some of these parameters cannot be defined, since they are outside from the image boarders.
Table A.1 summarizes the padding rules defined in the standard to provide always a value to each reconstructed sample. For the sake of completeness, in the table x denotes the row index, y the column one, and N_{cols} the total number of columns in the image.

Table A.1: Padding rules

Reconstructed Sample	**Position**	**Assigned value**
Ra	$x = 0,\ y = 0$	0
	$x \neq 0,\ y = 0$	Rb
Rb	$x = 0$	0
Rc	$x = 0$	0
	$x \neq 0,\ y = 0$	Rb of first pixel of previous row
Rd	$x = 0$	0
	$x \neq 0,\ y = N_{cols} - 1$	Rb

From these values, the context determines if the information of the current sample should be encoded in the *Regular* or *Run* mode (i.e., the way in which the reconstructed value associated with a pixel is computed varies depending on the selected encoding mode, so the procedure to compute this value is described in Sections A.0.2.3 and A.0.2.4).

A.0.2.3 Regular Mode

As shown in Figure 2.30, when the Regular encoding mode is selected, a predicted value (Px) for the current pixel is computed (*Prediction*). Px can be computed exploiting the following algorithm.
After Px is computed, the prediction shall be corrected. The correction depends on $SIGN$ (Section A.0.2.2) and the prediction correction value $C[context]$. The detailed procedure to perform the correction is reported in Algorithm 7.
In the previous algorithm, the $MAXVAL$ parameter is the maximum pixel value, so it is equal to $2^{bpp} - 1$, where bpp is the number of bit used to represent a pixel.

Algorithm 6 Sample prediction

```
if Rc ≥ max(Ra,Rb) then
    Px = min(Ra,Rb)
else
    if Rc ≤ min(Ra,Rb) then
        Px = max(Ra,Rb)
    else
        Px = Ra+Rb−Rc
    end if
end if
```

Algorithm 7 Prediction correction

```
if SIGN = 1 then
    Px = Px+C[context]
else
    Px = Px−C[context]
end if
if Px > MAXVAL then
    Px = MAXVAL
else
    if Px < 0 then
        Px = 0
    end if
end if
```

Using the corrected value of Px, the prediction error ($Errval$) can be computed following Algorithm 8 (*Prediction Error* task in Figure 2.30).

Algorithm 8 Prediction error

```
ErrVal = Ix−Px
if SIGN = 1 then
    ErrVal = −ErrVal
end if
```

If the near-lossless compression has been selected, the prediction error must be quantized.
As suggested in the C/C++ implementation of the JPEG-LS, the quantization process can be performed by using two LUTs: $qdiv$, and $qmul$. By pointing $qdiv$ with $Errval$ the first quantization step can be performed (the output value is called $qErrVal$). The quantization can be concluded by computing the $iqErrVal$ variable by pointing $qmul$ with $qErrVal$.
Eventually, the modulo reduction of $qErrVal$ is performed as shown below. At this point, the reconstructed sample associated with the current pixel (Rx (i.e., this value will become the Ra sample for the next processed pixel)) can be computed as reported in Algorithm 10.
After all these steps, the computed errors can be encoded to obtain the compressed bitstream,

Algorithm 9 $qErrVal$ modulo reduction

1: **if** $qErrVal < 0$ **then**
2: $qErrVal = qErrVal + qbeta$
3: **end if**
4: **if** $qErrVal \geq ceil\_half\_qbeta$ **then**
5: $qErrVal = qErrVal - qbeta$
6: **end if**

Algorithm 10 Sample reconstruction

1: **if** $SIGN = 1$ **then**
2: $Rx = Px - iqErrVal$
3: **else**
4: $Rx = Px + iqErrVal$
5: **end if**
6: **if** $Rx > MAXVAL$ **then**
7: $Rx = MAXVAL$
8: **else**
9: **if** $Rx < 0$ **then**
10: $Rx = 0$
11: **end if**
12: **end if**

associated with the current processed pixel, to be appended to the compressed image.
This encoding phase is performed by means of the Golomb coding. First, the Golomb coding variable k is computed (Algorithm 11) exploiting the A[0..364], and N[0..364] vectors.

Algorithm 11 k computation

1: **for** $k = 0 \rightarrow (N[context] << k) < A[context]$ **do**
2: **end for**

Then, the prediction error must be mapped on a non negative value (*Error Mapping* task in Figure 2.30), as reported in Algorithm 12, producing the two mapped errors $AbsErrVal$, and $MErrVal$.

Algorithm 12 Error mapping

1: **if** $qErrVal \geq ceil\_half\_qbeta$ **then**
2: $qErrVal = qErrVal - qbeta$
3: $absErrVal = -qErrVal$
4: $MErrVal = 2 * absErrVal - 1$
5: **else**
6: $absErrVal = qErrVal$
7: $MErrVal = 2 * qErrVal$
8: **end if**

Eventually, the Golomb coding can be performed and the current processed pixel can be compressed as reported in Algorithm 13 (*Error Encoding* task in Figure 2.30). In this algorithm the function $append\_zeros(n)$ appends n zeros to the output compressed image, the function $append\_bits(value, n)$ appends $value$ represented on n bits to the output compressed image, and & performs the bit-wise logical and operation.

Algorithm 13 Regular Golomb coding

1: $unary = MErrVal >> k$
2: **if** $unary < LIMIT$ **then**
3: $\quad append\_zeros(unary)$
4: $\quad append\_bits((1 << k) + (MErrVal \ \& \ ((1 << k) - 1)), (k + 1))$
5: **else**
6: $\quad append\_zeros(LIMIT)$
7: $\quad append\_bits((1 << qbpp) + MErrVal - 1)), (qbpp + 1))$
8: **end if**

Before concluding the regular encoding phase, the A, B, C, and N vectors must be update as shown in Algorithm 14.

A.0.2.4 Run Mode

If the local gradients are all equal to zero (for lossless coding), or their absolute values are less than or equal to $NEAR$ (for near-lossless coding), then the process enters in Run Mode.
In lossless coding, the encoder reads subsequent samples into Ix while Ix equals the reconstructed sample Ra at the beginning of the run, or the end of the current image line is encountered.
Instead, in near-lossless coding, if the absolute value of the difference between Ix and Ra is less than or equal to the allowed error ($NEAR$), the run continues.
The encoding of the run length is followed by the encoding of the last scanned sample (i.e. run interruption sample) in case the run is interrupted other than by the end of the current image line.
From this initial description, the run mode procedure can be split into three main steps: *Run scanning*, *Run-length coding*, and *Run interruption coding* (Figure 2.30).

A.0.2.5 Run scanning

The *Run scanning* step runs inside the image row associated with the current pixel until it finds a run interruption sample, as depicted in Algorithm 15. In this algorithm, the variable $EOLine$ is asserted when the end of an image run is reached, and the function $GetNextPixel$ return the pixel following the currently processed one.

Algorithm 14 A, B, C, and N updating

```
1: B[context] = B[context] + qmul[qErrVal]
2: A[context] = A[context] + AbsErrVal
3: if N[context] = RESET then
4:     N[context] = N[context] >> 1
5:     A[context] = A[context] >> 1
6:     B[context] = B[context] >> 1
7: end if
8: N[context] = N[context] + 1
9: if B[context] = −N[context] then
10:    if C[context] > MIN_C then
11:        C[context] = C[context] − 1
12:    end if
13:    B[context] = B[context] + N[context]
14:    if B[context] ≤ −N[context] then
15:        B[context] = −N[context] + 1
16:    end if
17: else
18:    if B[context] > 0 then
19:        if C[context] < MAX_C then
20:            C[context] = C[context] + 1
21:        end if
22:        if B[context] > 0 then
23:            B[context] = 0
24:        end if
25:    end if
26: end if
```

Moreover, it must be specified that, in the case of lossless compression, the control $|Ix-RUNval|$ $\leq NEAR$ is reduced to $Ix = RUNval$.

Algorithm 15 Run scanning

```
1: RUNval = Ra
2: RUNcount = 0
3: while |Ix − RUNval| ≤ NEAR do
4:     RUNcount = RUNcount + 1
5:     Rx = RUNval
6:
7:     if EOLine = 1 then
8:         break
9:     end if
10:    Ix = GetNextPixel()
11: end while
```

A.0.2.6 Run-length encoding

After the computation of Algorithm 15, the variable $RUNcnt$ is equal to the number of pixel representing the processed run. The value of $Runcnt$ must be encoded (*Run-length encoding* task in Figure 2.30) exploiting the block-MELCODE coding [165], an adaptation technique for Golomb code [80].
This encoding process exploits three variables: $melcorder$, $melcstate$, and $melclength$, that are initialized at the beginning of each image compression at 1, 0, and 0, respectively.
Algorithm 16 details the block-MELCODE coding operations. In this algorithm the function $append\_ones(n)$ appends n ones to the output compressed image.

Algorithm 16 Run-length encoding

```
1:  hits = 0
2:  while Runcnt >= melcorder do
3:      hits = hits + 1
4:      Runcnt = Runcnt − melcorder
5:      if melcstate < 32 then
6:          melcstate = melcstate + 1
7:          melclength = J[melcstate]
8:          melcorder = 2^melclength
9:      end if
10: end while
11: append_ones(hits)
12: if EOLine = 1 then
13:     if Runcnt > 0 then
14:         append_ones(1)
15:     end if
16:     return
17: end if
18: limit_reduced = melclength + 1
19: append_bits(Runcnt, limit_reduced)
20: if melcstate > 0 then
21:     melcstate = melcstate + 1
22:     melclength = J[melcstate]
23:     melcorder = 2^melclength
24: end if
```

It must be noted that if the the run is stopped by the end of a line, only the ones are appended to the output compressed image, and the *Run interruption coding* is skipped ($return$ at row 16 of Algorithm 16).

A.0.2.7 Run interruption coding

The basic concepts in the run interruption encoding are the same as those used to encode a new sample in the regular encoding mode (Section A.0.2.3), with the additional requirement that Ix

must differ from Ra by more than $NEAR$, otherwise the run would have continued.
The Run interruption coding phase is performed if the run is interrupted other than by the end of the image line. In this case, the run interruption sample must be encoded.
In this phase the value of the context (Q) is defined starting from the RI_type variable that is computed according to Algorithm 17.

Algorithm 17 RI_type computation

```
1: if |Ra − Rb| ≤ NEAR then
2:     RI_type = 1
3: else
4:     RI_type = 0
5:     Q = 365 + RI_type
6: end if
```

Moreover, RI_type is used to define the prediction value (xpr) and the prediction error ($ErrVal\_run$) of the run interruption sample (Algorithm 18).

Algorithm 18 xpr and ErrVal_run computation

```
1: if RI_type = 1 then
2:     xpr = Ra
3:
4:     ErrVal_run = Ix − xpr else
5:     xpr = Rb
6:     ErrVal_run = Ix − xpr
7:     if Rb < ra then
8:         ErrVal_run = −ErrVal_run
9:     end if
10: end if
```

As in the regular mode encoding, if the near-lossless compression has been selected, the prediction error must be quantized using the two LUTs: *qdiv*, and *qmul*, in order to obtain the *qErrVal_run* and the *iqErrVal_run* variables.
Moreover, as show in Algorithm 9, the modulo reduction of $qErrVal\_run$ is performed to obtain the $qErrVal\_final$ value.
At this point, the reconstructed sample associated with the current pixel (Rx) can be computed as reported in Algorithm 19.
In the run mode encoding, the Golomb coding variable *k*, as shown in Algorithm 20, is computed in a slightly different way w.r.t. the regular mode encoding.
Before computing the Golomb coding of the run interruption sample, the prediction error must be mapped, as reported in Algorithm 21, producing the two mapped errors $AbsErrVal\_run$, and $MErrVal\_run$.
Eventually, the Golomb coding can be performed and the current processed pixel can be com-

Algorithm 19 Sample reconstruction

```
if ((RI_type = 1)||(Rb ≥ Ra)) then
    Rx = xpr + iqErrVal_run
else
    Rx = xpr − iqErrVal_run
end if
if Rx > MAXVAL then
    Rx = MAXVAL
else
    if Rx < 0 then
        Rx = 0
    end if
end if
```

Algorithm 20 k computation

```
if RI_type = 1 then
    A_new = A[Q] + N[Q]/2
else
    A_new = A[Q]
end if
Nt = N[Q]
for k = 0 → Nt < A_new do
    Nt = Nt ∗ 2
end for
```

Algorithm 21 qErrVal_run mapping

```
oldmap = (k == 0 & qErrval_run & 2 ∗ B[Q] < Nt)
if qErrVal_run < 0 then
    MErrVal_run = −2 ∗ qErrVal_run − RI_type + oldmap
else
    MErrVal_run = 2 ∗ qErrVal_run − RI_type − oldmap
end if
AbsErrVal = (MErrVal + 1 − RI_type)/2
```

pressed as reported in Algorithm 22. It must be noted that, the variable $limit\_reduced$ is the one computed during the Run-length encoding (Algorithm 16).

Before concluding the run encoding of the current pixel, the value of the A, B, and N vectors must be update as shown in Algorithm 23. It must be noted, that differently from the regular mode encoding, the C vector is not defined and used in this encoding mode.

Algorithm 22 Run Golomb coding

1: $right\_limit = LIMIT - limit\_reduced$
2: $unary = MErrVal\_run >> k$
3: **if** $unary < right\_limit$ **then**
4: $\quad append\_zeros(unary)$
5: $\quad append\_bits((1 << k) + (MErrVal\_run \ \& \ ((1 << k) - 1)), (k+1))$
6: **else**
7: $\quad append\_zeros(right\_limit)$
8: $\quad append\_bits((1 << qbpp) + MErrVal - 1)), (qbpp + 1))$
9: **end if**

Algorithm 23 A, B, and N updating

1: **if** $qErrVal\_run < 0$ **then**
2: $\quad B[Q] = B[Q] + 1$
3: **end if**
4: $A[Q] = A[Q] + AbsErrVal$
5: **if** $N[Q] = RESET$ **then**
6: $\quad N[Q] = N[Q] >> 1$
7: $\quad A[Q] = A[Q] >> 1$
8: $\quad B[Q] = B[Q] >> 1$
9: **end if**
10: $N[Q] = N[Q] + 1$

APPENDIX B

LIST OF SYMBOLS AND ACRONYMS

Generic name	Description
$A[]$	367-cells vector saving the counters that store the accumulated prediction error magnitudes
$AbsErrVal$, $MErrVal$	Two mapped errors in the Regular-mode Encoding
$AbsErrVal\_run$, $MErrVal\_run$	Two mapped errors in the Run-mode Encoding
acc	Accumulator
$actual\_BW$	Current Histogram Bar Width
ADD Tree	Adders tree
$addr$	Address port
AHE	Adaptive Histogram Equalization
AHFE	Adaptive Harris Features Extractor
AIDI	Adaptive Image Denoiser Ip-core
alpha	Maximum pixel value in the image
α_H	light direction component along the horizontal axis
$\alpha_{P(x,y)}$	Angle between the x axis and the tangent surface associated to the currently processed pixel
α_V	light direction component along the vertical axis
AOCS	Attitude & Orbit Control System
append_bits(*value*,*n*)	Append to a bit string *value* represented on *n* bit
append_ones(*n*)	Append *n* ones to a bit string
append_zeros(*n*)	Append *n* zeros to a bit string
ASIC	Application Specific Integrated Circuit
ATB	Thales Alenia Space Avionic Testbench
ATHC	Adaptive Threshold Computation Module
AVS	Approximation of Viscosity Solution

Continue on next page

Table B.1 – Continue from previous page

Acronym or Symbol	Meaning
B	Buffer
b	Baseline of the stereo camera
B[]	367-cells vector storing the bias values used to update the prediction correction variable
bpp	bit-per-pixel
BRAM	Block-Random Access Memory
BW	Bar Width
BW_flag	Flag signal asserted when the Histogram Bar width is greater than the input Histogram Bar width threshold
C[]	365-cells vector storing the prediction correction value
CBM	Cruise Balance Mass
CC	Cross-Correlation
CCC	Clock Conditioning Circuitry
CCD	Charge Couple Device
CLAHE	Contrast Limited Adaptive Histogram Equalization
$ClassMapvLUT$	Look-Up Table to define the final value of the context
CLB	Configurable Logic Block
CLK	Clock signal
CLR	Clear signal
CM	Correct Matches
CMT	Clock Management Tile
$context$	Regular-mode encoding context
CS	Computation Stage
$Curr\_EF$	Current overall number of extracted features
$Current\_TH$	Current threshold value
D	Pixel depth
data_in	Input data port
data_out	Output data port
DCM	Digital Clock Manager
DDR	Double Data Rate
δ	Tolerance on the different between the number of extracted and target features
Δ_H	Slope contribution provided by the currently processed pixel
δ_x	Haar wavelet response along the horizontal direction
δ_y	Haar wavelet response along the vertical direction

Continue on next page

Table B.1 – Continue from previous page

Acronym or Symbol	Meaning
$Det(X)$	Determinant of matrix X
D-FF	D-type Flip Flop
DFT	Design For Testability
$Disp$	Displacement
DMA	Direct Memory Access
DoG	Difference of Gaussian
DPR	Dynamic Partial Reconfiguration
DRD	Design Requirement Document
DRL	Design Requirement List
DSP	Digital Signal Processor
Dx	Image local gradient (x can be equal to 1, 2, and 3)
EBDM	Ejectable Mass Balance Device
ECC	Error Correcting Code
EDAC	Error Detection And Correction
EDL	Entry Descent and Landing
en	Enable signal
$EOLine$	Flag asserted when the end of an image line is reached
$ErrVal$	Prediction Error in the Regular-mode encoding
$ErrVal\_run$	Prediction Error in the Run-mode encoding
ESA	European Space Agency
ESTEC	European Space and Technology Centre
FB	Features Buffer
FEM	Features Extraction and Matching
FEMIP	Features Extraction and Matching IP-core
FFT	Fast Fourier Transform
$FI(x,y)$	Filtered pixel value
Field Programmable Gate Array	FPGA
FIFO	First In First Out
FoV	Field of View
fps	frame-per-second
FSM	Finite State Machine
FTMR	Functional Triple Modular Redundancy
FVE	Filter Variance Estimator
$G(x,y,\sigma)$	Gaussian blurring function
Gap	Difference between the number of extracted and target features

Continue on next page

Table B.1 – Continue from previous page

Acronym or Symbol	Meaning
$GetNextPixel()$	Return the pixel value following the currently processed one
GNC	Guidance Navigation and Control
GRM	General Routing Matrix
G_x	Sobel kernel for detecting edges in the horizontal direction
G_y	Sobel kernel for detecting edges in the vertical direction
H	Image height
$H(x, y)$	Combination of the slope contributions along the horizontal and vertical axis
HA	Histogram address signal
HA_done	Flag signal asserted when the Histogram Analysis has been successfully computed
HAD	Flag signal asserted when the Histogram Analysis has been successfully computed
Hazcam	Hazard avoidance camera
HB	Histogram Bar
HB_addr	Histogram Bar addressing signal
HB_sum	Histogram Bars sum
HB_sum_done	Flag signal asserted when the Histogram Bars sum has been successfully computed
HDL	Hardware Description Language
HE	Histogram Equalization
HE_done	Flag signal asserted when the Histogram Equalization has been successfully computed
HED	Flag signal asserted when the Histogram Equalization has been successfully computed
hist_addr	Histogram address signal
hist_done	Flag signal asserted when the Image Histogram has been successfully computed
$hits$	number of loop performed during the Run-length encoding
HS	Histogram Stretching
HS_done	Flag signal asserted when the Histogram Stretching has been successfully computed
HSD	Flag signal asserted when the Histogram Stretching has been successfully computed
HW	Histogram Width

Continue on next page

Table B.1 – Continue from previous page

Acronym or Symbol	Meaning
HW_flag	Flag signal asserted when the Histogram width is greater than the input Histogram width threshold
$I(x,y)$	Input image pixel value
I/O	Input/Output
ICAP	Internal Configuration Port
ICD	Interface Control Document
IET	Image Enhancement Technique
$II(x,y)$	Integral image pixel value
I_{max}	Maximum brightness value in the input image
I_{min}	Minimum brightness value in the input image
IMU	Inertial Measurement Unit
IP	Intellectual Property
$iqErrVal$	Final value of the quantized prediction error in the Regular-mode encoding
$iqErrVal\_run$	Final value of the quantized prediction error in the Run-mode encoding
IRIS	Interface Region Imaging Spectrograph
I_x	Currently compressed pixel
J[]	32-cells constant vector for the Block-MELCODE coding
JTAG	Joint Test Action Group
k	Golomb coding variable
KFS	Kernel Factor Selector
$L(x,y,\sigma)$	Gaussian blurred pixel
LAPU	Live At Power-Up
LE	Logic Element
LEO	Low Earth Orbit
LIMIT	Length parameter used during the Golomb coding
$limit\_reduced$	$melclength$ value increased by one
LoG	Laplacian of Gaussian
$LowTH$	Lower threshold bound
LUT	Look-Up Table
LUT-3	3-input LUT
LUT4	4-input LUT
LUT6	6-input LUT
LVE	Local Variance Estimator

Continue on next page

Table B.1 – Continue from previous page

Acronym or Symbol	Meaning
L_x	Image derivative along the horizontal direction
L_y	Image derivative along the vertical direction
$m(x,y)$	Gradient magnitude
MAC	Multiply-And-Accumulate
MARDI	MARs Descent Imager
max	Maximum Histogram Bar value signal
$max(A,B)$	Return the maximum value between A and B
max_BW	Maximum Histogram Bar width value
max_HB	Maximum Histogram Bar value
MAXVAL	Maximum pixel value in the image
ME	Motion Estimation
$melcorder, melcstate, melclength$	Block-MELCODE coding variables
min	Minimum Histogram Bar value signal
$min(A,B)$	Return the minimum value between A and B
min_HB	minimum Histogram Bar value
MISSE	Materials International Space Station Experiment
MLE	Mars Lander Engine
Mod C2	Module computing the complement 2 absolute value
μ	Gradient orientation
MUL/ADD Tree	Multipliers and adders tree
MUX	Multiplexer
N	Laplacian kernel
n	Normal to the tangent surface to an observed object point
$N(x,y)$	Second Moment matrix
$N[]$	367-cells vector storing the counters of the frequency of occurrence of computed pixel contexts
N_cell	Number of image cells
$N\_Sobel\_features$	Number of features extracted exploiting the Sobel extractor
$N\_target\_features$	Number of target features
NAVCAM	NAVigation CAMera
NCC	Normalized Cross-Correlation
$NEAR$	Allowed error for near-lossless coding
NEM	Number of Extracted Matches
new_TH	New threshold value
NF	Number of extracted features

Continue on next page

Table B.1 – Continue from previous page

Acronym or Symbol	Meaning
$NF[]$	Matrix defining the number of extracted features from the image cells
NMS	Non-Maxima Suppression
NPAL	Navigation for Planetary Approach and Landing
NRE	Non-Recurrent Engineering
NTH	New threshold value
NVE	Noise Variance Estimator
$Offset$	Correction value for the new threshold value
OTF	Overall number of expected features per frame
OTP	One-Time-Programmable
$P(x,y)$	Currently processed pixel
PDE	Partial Differential Equation
PLL	Phase Locked Loop
$previous\_HB$	Previous Histogram Bar value
PSNR	Peak Signal to Noise Ratio
Px	Predicted compressed pixel value
Q	Run-mode encoding context
Q_sum	Sum of the quantized contexts
$qBeta, ceil\_half\_qBeta$	Variables for the Error Mapping
$qbpp$	Number of bit required to represent a mapped error
$qdiv$	Look-Up Table for the first step of the prediction error quantization
$qErrVal$	Quantized prediction error after the first quantization step in the Regular-mode encoding
$qErrVal\_final$	qErrval_run with the modulo reduction applied
$qErrVal\_run$	Quantized prediction error after the first quantization step in the Run-mode encoding
$qmul$	Look-Up Table for the second step of the prediction error quantization
Qx	Quantized context of the image local gradient (x can be equal to 1, 2, and 3)
R	Register
R	Reconfiguration task
$R(x,y)$	Corner Response function
RB	Row Buffer
$real\_HW$	Actual Histogram Bar width

BIBLIOGRAPHY

[1] Asic design methodology. In *Advanced ASIC Chip Synthesis Using Synopsys Design Compiler Physical Compiler and PrimeTime*, pages 1–17. Springer US, 2002.

[2] Suneja A. and Ekta W. A conceptual study on image matching techniques. *Global Journal of Computer Science and Technology*, 10(12):83–88, 2010.

[3] L. Abada and S. Aouat. Solving the perspective shape from shading problem using a new integration method. In *Proc. of Science and Information Conference (SAI)*, pages 416–422, 2013.

[4] European Space Agency. Gaia mission, [Last Access: December 2014]. `http://sci.esa.int/gaia/`.

[5] European Space Agency. Hipparcos mission, [Last Access: December 2014]. `http://www.esa.int/Our_Activities/Space_Science/Hipparcos_overview`.

[6] European Space Agency. Inside Gaia's billion pixel camera, [Last Access: December 2014]. `http://sci.esa.int/gaia/53281-inside-gaias-billion-pixel-camera/`.

[7] European Space Agency. Rosetta mission, [Last Access: December 2014]. `http://www.esa.int/Our_Activities/Space_Science/Rosetta`.

[8] W.J. Alldridge. Aocs technology [attitude and orbit control subsystem]. In *Proc. of IEEE Colloquium on UK Interest in Horizon 2000 - The ESA Space Science Programme*, pages 6/1–6/9, 1990.

[9] A.M. Alsuwailem and S.A. Alshebeili. A new approach for real-time histogram equalization using FPGA. In *Proc. of 2005 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pages 397–400, 2005.

[10] Altera . *White Paper: Increase Performance in Video and Image Processing Applications With FPGA Integration*, 2008.

[11] Analog Devices Inc. *Fundamental on Phase Locked Loops (PLLs)*, 2008. Rev. 0, MT-086 Tutorial.

[12] Harry John Philip Arnold. *William Henry Fox Talbot: Pioneer of Photography and Man of Science*. Hutchinson Benham, first edition, 1977.

[13] A. Avizienis. The n-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, SE-11(12):1491–1501, 1985.

[14] Nicholas Ayache. *Artificial vision for mobile robots: stereo vision and multisensory perception*. MIT Press, 1991.

[15] Jong Sue Bae and Taek Lyul Song. A survey on image enhancement methods. *International Journal of Control, Automation, and Systems (IJCAS)*, 6(3):413–423, 2008.

[16] N. Battezzati, S. Colazzo, M. Maffione, and L. Senepa. SURF algorithm in FPGA: A novel architecture for high demanding industrial applications. In *Proc. of Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 161–162, 2012.

[17] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. *Computer Vision and Image Understanding (CVIU)*, 110:346–359, 2008.

[18] P. Beaudet. Rotationally invariant image operators. In *Proc. of 4th International Joint Conference on Patter Recognition*, pages 579–583, 1978.

[19] A. Benedetti and P. Perona. Real-time 2D feature detection on a reconfigurable computer. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVP)*, pages 586–593, 1998.

[20] Robin Biesbroek. The e.Deorbit CDF Study - A design study for the safe removal of a large space debris. In *Proc. of 6th International Association for the Advanced of Space Safety (IAASS) Conference*, 2013.

[21] R. Bonamy, D. Chillet, S. Bilavarn, and O. Sentieys. Power consumption model for partial and dynamic reconfiguration. In *Proc. of International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2012.

[22] Charles Boncelet. Image noise models. In Al Bovik, editor, *Handbook of Image and Video Processing (Second Edition)*, pages 397 – 409. Academic Press, 2005.

[23] Ch. Bonnal and W. Naumann. Ariane debris mitigation measures: Past and future. *Acta Astronautica*, 40(2–8):275–282, 1997.

[24] Hephaestus Books. *Articles on History of Photography, Including: Camera Obscura, Timeline of Photography Technology, Calotype, William Fox Talbot, Daguerreotype, Lacock Abbey, Land Camera, National Media Museum, Group F/64, Louis Daguerre, Cyanotype*. Hephaestus Books, paperback - trade edition, 2011.

[25] D. Bouris, A. Nikitakis, and I. Papaefstathiou. Fast and efficient FPGA-based feature detection employing the SURF algorithm. In *Proc. of 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 3–10, 2010.

[26] C. Cabani and W.J. MacLean. A proposed pipelined-architecture for FPGA-based affine-invariant feature detectors. In *Proc. of Computer Vision and Pattern Recognition Workshop (CVPRW)*, pages 121–126, 2006.

[27] Qi Cai, Li Song, Guichun Li, and Nam Ling. Lossy and lossless intra coding performance evaluation: Hevc, h.264/avc, jpeg 2000 and jpeg ls. In *Proc. of Signal Information Processing Association Annual Summit and Conference (APSIPA ASC) Asia-Pacific*, pages 1–9, 2012.

[28] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[29] Ju Yong Chang, Kyoung Mu Lee, and Sang Uk Lee. Shape from shading using graph cuts. In *Proc. of International Conference on Image Processing (ICIP)*, volume 1, pages I–421–4 vol.1, 2003.

[30] Jie Chen, Li hui Zou, Juan Zhang, and Li hua Dou. The comparison and application of corner detection algorithms. *Journal of Multimedia*, 4(6), 2009.

[31] Yong-Sheng Chen, Yi-Ping Hung, and Chiou-Shann Fuh. Fast block matching algorithm based on the winner-update strategy. *IEEE Transactions on Image Processing*, 10(8):1212–1222, 2001.

[32] Yang Cheng and Adnan Ansar. Landmark based position estimation for pinpoint landing on mars. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pages 4470–4475, 2005.

[33] Yang Cheng and JK Miller. Autonomous landmark based spacecraft navigation system. In *Proc. of AAS/AIAA Astrodynamics Specialist Conference*, 2003.

[34] Yang Cheng and J.K. Miller. Autonomous landmark based spacecraft navigation system. In *Proc. of AAS/AIAA Astrodynamics Specialist Conference*, 2003.

[35] Chi Kin Chow and Shiu Yin Yuen. Recovering shape by shading and stereo under lambertian shading model. *International Journal of Computer Vision*, 85(1):58–100, 2009.

[36] Craig L Cole and John L Crassidis. Fast star-pattern recognition using planar triangles. *Journal of guidance, control, and dynamics*, 29(1):64–71, 2006.

[37] Hansel A. Collins and Ronald E. Nikel. High-speed source synchronous interface design. *Insight*, 4(3):19–21, 1999.

[38] Carlos Colodro-Conde, F Javier Toledo-Moreo, Rafael Toledo-Moreo, J Javier Martínez-Álvarez, Javier Garrigós Guerrero, and J Manuel Ferrández-Vicente. Evaluation of stereo correspondence algorithms and their implementation on FPGA. *Journal of Systems Architecture*, 60(1):22–31, 2014.

[39] Altera Corporation. *Video and Image Processing Suite User Guide*. Altera Corporation, 2014.

[40] Microsemi Corporation. RT ProASIC3 FPGAs, [Last Access: December 2014]. http://www.microsemi.com/products/fpga-soc/radtolerant-fpgas/rt-proasic3.

[41] Microsemi Corporation. RTAX-S/SL FPGAs, [Last Access: December 2014]. http://www.microsemi.com/products/fpga-soc/radtolerant-fpgas/rtax-s-sl#overview.

[42] QSI Corporation. Understanding CCD Read Noise, [Last Access: December 2014]. www.qsimaging.com/ccd_noise.html.

[43] Pascal Daniel and Jean denis Durou. From deterministic to stochastic methods for shape from shading. In *Proc. of 4th Asian Conference on Computer. Vision*, pages 187–192, 2000.

[44] Alan J. Danker and Azriel Rosenfeld. Blob detection by relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3(1):79–92, 1981.

[45] Leonard David. How Huffing and Puffing Could Remove Space Junk, [Last Access: December 2014]. http://www.space.com/15178-space-junk-removal-spade.html.

[46] Larry S. Davis. A survey of edge detection techniques. *Computer Graphics and Image Processing*, 4(3):248 – 270, 1975.

[47] G. Deng and L.W. Cahill. An adaptive gaussian filter for noise reduction and edge detection. In *Proc. of Nuclear Science Symposium and Medical Imaging Conference*, pages 1615 – 1619 vol.3, 1993.

[48] G. Deng and L.W. Cahill. An adaptive gaussian filter for noise reduction and edge detection. In *Proc. of Nuclear Science Symposium and Medical Imaging Conference*, pages 1615 – 1619 vol.3, 1993.

[49] S. Di Carlo, G. Gambardella, M. Indaco, D. Rolfo, G. Tiotto, and P. Prinetto. An area-efficient 2-D convolution implementation on FPGA for space applications. In *Proc. of 6th International Design and Test Workshop (IDT)*, pages 88 –92, 2011.

[50] S. Di Carlo, G. Gambardella, M. Indaco, D. Rolfo, G. Tiotto, and P. Prinetto. An area-efficient 2-D convolution implementation on FPGA for space applications. In *Proc. of 6th International Design and Test Workshop (IDT)*, pages 88 – 92, 2011.

[51] S. Di Carlo, G. Gambardella, P. Lanza, P. Prinetto, D. Rolfo, and P. Trotta. SAFE: a Self Adaptive Frame Enhancer FPGA-based IP-core for real-time space applications. In *Proc. of 7th International Design and Test Workshop (IDT)*, 2012.

[52] S. Di Carlo, G. Gambardella, P. Prinetto, D. Rolfo, and P. Trotta. Sa-femip: A self-adaptive features extractor and matcher ip-core based on partially reconfigurable FPGAs for space applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, PP(99):1–1, 2014.

[53] S. Di Carlo, G. Gambardella, P. Prinetto, D. Rolfo, P. Trotta, and P. Lanza. FEMIP: A high performance FPGA-based features extractor and matcher for space applications. In *Proc. of 23rd International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, 2013.

[54] S. Di Carlo, P. Prinetto, D. Rolfo, and P. Trotta. AIDI: An adaptive image denoising FPGA-based IP-core for real-time applications. In *Proc. of NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 99–106, 2013.

[55] Stefano Di Carlo, Paolo Prinetto, Daniele Rolfo, and Pascal Trotta. A fault injection methodology and infrastructure for fast single event upsets emulation on Xilinx SRAM-based FPGAs. In *Proc. of IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 159–164, 2014.

[56] R. Dobai and L. Sekanina. Image filter evolution on the xilinx zynq platform. In *Proc. of NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 164–171, 2013.

[57] Y. Dufournaud, C. Schmid, and R. Horaud. Matching images with different resolutions. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 612–618 vol.1, 2000.

[58] M. Dunstan, S. Parkes, and S. Mancuso. Visual navigation chip for planetary landers. In *Proc. of Conference on Data Systems In Aerospace (DASIA)*, pages 1–7, 2005.

[59] M. Dunstan and M. Souyri. The FEIC development for NPAL project: A core image processing chip for smart landers navigation applications. In *MicroElectronics Presentation Days, ESA/ESTEC*, 2004.

[60] Jean-Denis Durou, Maurizio Falcone, and Manuela Sagona. A survey of numerical methods for shape from shading. *Rapport de recherche*, 2, 2004.

[61] Jean-Denis Durou, Maurizio Falcone, and Manuela Sagona. Numerical methods for shape-from-shading: A new survey with benchmarks. *Computer Vision and Image Understanding (CVIU)*, 109(1):22–43, 2008.

[62] Jean-Denis Durou and Didier Piau. Ambiguous shape from shading with critical points. *Journal of Mathematical Imaging and Vision*, 12(2):99–108, 2000.

[63] EADS Astrium. GR-CPCI-XC4V LEON PCI Virtex 4 development board - Product sheet, [Last Access: December 2014]. http://www.scribd.com/doc/18759944/EADS-Navigation-for-Planetary-Approach-Landing.

[64] M.M El-gayar, H. Soliman, and N. Meky. A comparative study of image low level feature extraction algorithms. *Egyptian Informatics Journal*, 14(2):175–181, 2013.

[65] Elsevier. International Journal of Applied Earth Observation and Geoinformation, [Last Access: December 2014]. http://www.journals.elsevier.com/international-journal-of-applied-earth-observation-and-geoinformation/.

[66] ESA. Sentinel-1 orbiter, [Last Access: December 2014]. http://www.esa.int/Our_Activities/Observing_the_Earth/Copernicus/Sentinel-1.

[67] European Space Agency (ESA). Launchers, [Last Access: December 2014]. http://www.esa.int/Our_Activities/Launchers/Ariane_42.

[68] European Space Agency (ESA). *Requirements on Space Debris Mitigation for ESA Projects*, 2009.

[69] European Space Agency (ESA). Space in images - MISSION: MARS EXPRESS, [Last Access: December 2014]. http://www.esa.int/spaceinimages/Missions/Mars_Express.

[70] Fairchild Imaging Corporation. *CCD424 1024 x 1024 Pixel Image Area Split Frame Transfer CCD Sensor*, 2002.

[71] D. Fay, A. Shye, S. Bhattacharya, D.A. Connors, and S. Wichmann. An adaptive fault-tolerant memory system for FPGA-based architectures in the space environment. In *Proc. of 2nd NASA/ESA Conference onAdaptive Hardware and Systems (AHS)*, pages 250–257, 2007.

[72] P.D. Ferguson, T. Arslan, A.T. Erdogan, and A. Parmley. Evaluation of contrast limited adaptive histogram equalization (CLAHE) enhancement on a FPGA. In *Proc. of 8th IEEE International SoC Conference*, pages 119–122, 2008.

[73] Robert Fisher. CVonline: Image Databases, [Last Access: December 2014]. http://homepages.inf.ed.ac.uk/rbf/CVonline/Imagedbase.htm.

[74] Y. Fouda and K. Ragab. An efficient implementation of normalized cross-correlation image matching based on pyramid. In *Proc. of International Joint Conference on Awareness Science and Technology and Ubi-Media Computing (iCAST-UMEDIA)*, pages 98–103, 2013.

[75] Jiri Gaisler. A portable and fault-tolerant microprocessor based on the sparc v8 architecture. In *Proc. of International Conference on Dependable Systems and Networks (DSN)*, pages 409–415. IEEE, 2002.

[76] Gaisler Research. *Functional Triple Modular Redundancy (FTMR) - VHDL Design Methodology for Redundancy in Combinatorial and Sequential Logic*, 2002. Ver. 0.2 , FPGA-003-01.

[77] Gaisler Research. *Suitability of reprogrammable FPGAs in space applications* , 2002. Ver. 0.4, FPGA-002-01 Feasibility Report.

[78] Gaisler Research AB. Navigation for Planetary Approach and Landing - Final Report, [Last Access: December 2014]. `http://www.pender.ch/docs/GR-CPCI-XC4V_product_sheet.pdf`.

[79] S. Ghosh. *Hardware description languages: concepts and principles*. IEEE Computer Society Press, 2001.

[80] S. Golomb. Run-length encodings (corresp.). *IEEE Transactions on Information Theory*, 12(3):399–401, 1966.

[81] R.C. González and R.E. Woods. *Digital Image Processing*. Pearson/Prentice Hall, 2008.

[82] R.C. González and R.E. Woods. *Digital Image Processing*. Pearson/Prentice Hall, 2008.

[83] Mentor Graphics. *ModelSim User Manual - Software Version 10.1c*. Mentor Graphics, 2012.

[84] Daniel Gregory. Space Debris Elimination (SpaDE), [Last Access: December 2014]. `http://www.nasa.gov/directorates/spacetech/niac/gregory_space_debris_elimination.html`.

[85] S. Habinc. Suitability of reprogrammable FPGAs in space applications - feasibility report. Technical report, Gaisler Research, 2002.

[86] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. of 4th Alvey Vision Conference*, pages 147–151, 1988.

[87] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[88] Hasselblad. History, [Last Access: December 2014]. `http://www.hasselbladusa.com/about-hasselblad/history.aspx`.

[89] Hasselblad. Space Cameras, [Last Access: December 2014]. `http://www.hasselbladusa.com/about-hasselblad/hasselblad-in-space/space-cameras.aspx`.

[90] I. Herrera-Alzu and M. Lopez-Vallejo. Design techniques for Xilinx Virtex FPGA configuration memory scrubbers. *IEEE Transactions on Nuclear Science*, 60(1):376–385, 2013.

[91] B. K.P. Horn. Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. Technical report, 1970.

[92] Berthold K. P. Horn and Michael J. Brooks, editors. *Shape from Shading*. MIT Press, 1989.

[93] Paul G. Howard and Jeffrey Scott Vitter. Fast and efficient lossless image compression. In *Proc. of Data Compression Conference (Snowbird)*, pages 351–360, 1993.

[94] Jeremy Hsu. Solar Sails Could Clean Up Space Junk, [Last Access: December 2014]. `http://www.space.com/8296-solar-sails-clean-space-junk.html`.

[95] Karishma S. Inamdar, Hari Shankar, and Ishani Salari. Space debris removal system. *International Journal of Emerging Technology and Advanced Engineering (IJETAE)*, 3(11):126–129, 2013.

[96] Sun Microsystems Inc. *Sun Enterprise 420R Server OwnerâĂŹs Guide*, 1999.

[97] Xilinx Inc. TMRTool, [Last Access: December 2014]. `http://www.xilinx.com/ise/optional_prod/tmrtool.htm`.

[98] Xilinx Inc. Video and Image Processing Pack, [Last Access: December 2014]. `http://www.xilinx.com/products/intellectual-property/ef-di-vid-img-ip-pack.html#overview`.

[99] Xilinx Inc. Xilinx All Programmable, [Last Access: December 2014]. `http://www.xilinx.com/about/index.htm`.

[100] Inter-Agency Space Debris Coordination Committee (IADC). *Key Definitions of the Inter-Agency Space Debris Coordination Committee (IADC) - IADC-13-02*, 2013.

[101] International Telecommunication Union. Itu-t.87 information technology - lossless and near-lossless compression of continuous-tone still images - baseline. 1998.

[102] International Telecommunication Union. T.87 : Information technology - Lossless and near-lossless compression of continuous-tone still images - Baseline, [Last Access: December 2014]. `https://www.itu.int/rec/T-REC-T.87-199806-I/en`.

[103] ITRS. 2013 International Technology Roadmap for Semiconductors, [Last Access: December 2014]. `http://www.itrs.net/Links/2013ITRS/Summary2013.htm`.

[104] Varghese J. Literature survey on image filtering techniques. *International Journal of Engineering Research & Technology (IJERT)*, 2(6):1672–1676, 2013.

[105] Jameel Hussein and Gary Swift. *Mitigating Single-Event Upsets* , 2012. Ver. 1.0, WP365.

[106] E. Jamro, M. Wielgosz, and K. Wiatr. FPGA implementaton of strongly parallel histogram equalization. In *Proc. of 10th Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 1–6, 2007.

[107] Japanese Aerospace Exploration Agency. Report on Space Debris Related Activities in Japan, [Last Access: December 2014]. `http://www.unoosa.org/pdf/natact/sdnps/2010/sd2011-japan.pdf`.

[108] A. Joginipelly, A. Varela, D. Charalampidis, R. Schott, and Z. Fitzsimmons. Efficient FPGA implementation of steerable gaussian smoothers. In *Proc. of 44th Southeastern Symposium on System Theory (SSST)*, pages 78 – 82, 2012.

[109] W. Vernon Jones. Aspirations and outlook for NASA cosmic ray research on balloons and in space. *Journal of the Physical Society of Japan*, 78:101–107, 2009.

[110] Marshall H Kaplan. Space debris realities and removal. In *Proc. of Improving Space Operations Workshop (SOSTC)*, 2010.

[111] Akira Kato, L Monika Moskal, Peter Schiess, Mark E Swanson, Donna Calhoun, and Werner Stuetzle. Capturing tree crown formation through implicit surface reconstruction using airborne lidar data. *Remote Sensing of Environment*, 113(6):1148–1162, 2009.

[112] D. Kessler. Collisional cascading: The limits of population growth in low earth orbit. *Advances in Space Research*, 11(12):63–66, 1991.

[113] Byung-Soo Kim, Seungkyu Baek, Dong-Sun Kim, and Duck-Jin Chung. A high performance fully pipeline jpeg-ls encoder for lossless compression. *IEICE Electronics Express*, 10(12):1–6, 2013.

[114] Ron Kimmel, Kaleem Siddiqi, BenjaminB. Kimia, and AlfredM. Bruckstein. Shape from shading: Level set propagation and viscosity solutions. *International Journal of Computer Vision*, 16(2):107–133, 1995.

[115] Kodak. George Eastman, [Last Access: December 2014]. `http://www.kodak.com/ek/US/en/Our_Company/History_of_Kodak/George_Eastman.htm`.

[116] Kodak. The first brownie camera, [Last Access: December 2014]. `http://www.kodak.com/US/en/corp/features/brownieCam/`.

[117] K. Kokufuta and T. Maruyama. Real-time processing of local contrast enhancement on FPGA. In *Proc. of 19th International Conference on Field Programmable Logic and Applications (FPL)*, pages 288–293, 2009.

[118] M. Kolesnik and G. Baratoff. 3D interpretation of sewer circular structures. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1453–1458 vol.2, 2000.

[119] Hui Kong, H.C. Akakin, and S.E. Sarma. A generalized laplacian of gaussian filter for blob detection and its applications. *IEEE Transactions on Cybernetics*, 43(6):1719–1733, 2013.

[120] K. Kyriakoulakos and D. Pnevmatikatos. A novel SRAM-based FPGA architecture for efficient tmr fault tolerance support. In *Proc. of International Conference on Field Programmable Logic and Applications (FPL)*, pages 193–198, 2009.

[121] Craig L., Cole, and John L. Crassidis. Fast star-pattern recognition using planar triangles. *Journal of Guidance, Control, and Dynamics*, 29(1):64–71, 2006.

[122] Ruan Lakemond, Clinton B. Fookes, and Sridha Sridharan. Affine adaptation of local image features using the hessian matrix. In *Proc. of IEEE International Conference On Advanced Video and Signal Based Surveillance*, 2009.

[123] R. Lakshmanan, M.S. Nair, M. Wilscy, and R. Tatavarti. Automatic contrast enhancement for low contrast images: A comparison of recent histogram based techniques. In *Proc. of 1st International Conference on Computer Science and Information Technology (ICCSIT)*, pages 269–276, 2008.

[124] F.-X. Lapalme, A. Amer, and Chunyan Wang. FPGA architecture for real-time video noise estimation. In *Proc. of International Conference on Image Processing*, pages 3257 – 3260, 2006.

[125] G. Lazaridis and M. Petrou. Image registration using the Walsh transform. *IEEE Transactions on Image Processing*, 15:2343–2357, 2006.

[126] F. Lima, L. Carro, and R. Reis. Designing fault tolerant systems into SRAM-based FPGAs. In *Proc. of Design Automation Conference*, pages 650–655, 2003.

[127] E. Loupias, N. Sebe, S. Bres, and J.-M. Jolion. Wavelet-based salient points for image retrieval. In *Proc. of International Conference on Image Processing*, volume 2, pages 518–521 vol.2, 2000.

[128] David G. Lowe. Object recognition from local scale-invariant features. In *Proc. of International Conference on Computer Vision - Volume 2*, 1999.

[129] Ching-Hsi Lu, Hong-Yang Hsu, and Lei Wang. A new contrast enhancement technique implemented on FPGA for real time image processing. In *Proc. of 5th International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*, pages 542–545, 2009.

[130] Ramkumar M. and Karthikeyan.B. A survey on image enhancement methods. *International Journal of Engineering Research & Technology (IJERT)*, 5(2):960–962, 2013.

[131] Mohammadreza Madi. Cleaning up Earth's orbit: A Swiss satellite tackles space debris, [Last Access: December 2014]. `http://actu.epfl.ch/news/cleaning-up-earth-s-orbit-a-swiss-satellite-tack-2/`.

[132] D. Marr and E. Hildreth. Theory of edge detection. In *Proc. of the Royal Society*, pages 187–217, 1980.

[133] Mars Science Laboratory. Curiosity Rover, [Last Access: December 2014]. `http://mars.jpl.nasa.gov/multimedia/images/`.

[134] MATLAB. *Fixed-Point Designer User's Guide - R2014b*. The MathWorks Inc., 2014.

[135] MATLAB. *Image Processing Toolbox User's Guide - R2014b*. The MathWorks Inc., 2014.

[136] T. Matsubara, V.G. Moshnyaga, and K. Hashimoto. A FPGA implementation of low-complexity noise removal. In *Proc. of 17th International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 255 – 258, 2010.

[137] C.R. McBryde and E.G. Lightsey. A star tracker design for cubesats. In *Proc. of IEEE Aerospace Conference*, pages 1–14, 2012.

[138] N. Merhav, G. Seroussi, and M.J. Weinberger. Optimal prefix codes for sources with two-sided geometric distributions. *IEEE Transactions on Information Theory*, 46(1):121–135, 2000.

[139] Microsemi Corporation. *Radiation-Tolerant ProASIC3 Low-Power Space-Flight FPGAs Datasheet*, 2012. Rev. 5, 51700107-5/9.12.

[140] Microsemi Corporation. *CoreEDAC - Handbook*, 2013. Ver. 2.5, 50200143-4/11.13.

[141] Microsemi Corporation. *RTAX-S/SL and RTAX-DSP Radiation-Tolerant FPGAs Datasheet*, 2013. Rev. 16, 5172169-16/1.13.

[142] Jonathan Missel and Daniele Mortari. Optimization of debris removal path for tamu sweeper. *Advances in the Astronautical Sciences*, 143(1):935–945, 2012.

[143] J. Mora, A. Gallego, A. Otero, E. de la Torre, and T. Riesgo. Noise-agnostic adaptive image filtering without training references on an evolvable hardware platform. In *Proc. of Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 182–189, 2013.

[144] J. Mora, A. Gallego, A. Otero, B. Lopez, E. de la Torre, and T. Riesgo. A noise-agnostic self-adaptive image processing application based on evolvable hardware. In *Proc. of Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 351–352, 2013.

[145] K.S. Morgan, D.L. McMurtrey, B.H. Pratt, and M.J. Wirthlin. A comparison of TMR with alternative fault-tolerant design techniques for FPGAs. *IEEE Transactions on Nuclear Science*, 54(6):2065–2072, 2007.

[146] A.I. Mourikis, N. Trawny, S.I. Roumeliotis, A.E. Johnson, A. Ansar, and L. Matthies. Vision-aided inertial navigation for spacecraft entry, descent, and landing. *IEEE Transactions on Robotics*, 25(2):264–280, 2009.

[147] Ibrahim Nahhas and Martin Drahansky. Analysis of block matching algorithms with fast computational and winner-update strategies. *International Journal of Signal Processing, Image Processing & Pattern Recognition*, 6(3), 2013.

[148] National Aeronautics and Space Administration (NASA). *Process for Limiting Orbital Debris*, 2011.

[149] National Aeronautics and Space Administration (NASA). Apollo 11 mission, [Last Access: December 2014]. `http://www.nasa.gov/mission_pages/apollo/missions/apollo11.html`.

[150] National Aeronautics and Space Administration (NASA). Curiosity mission, [Last Access: December 2014]. `http://mars.jpl.nasa.gov/msl/mission/overview/`.

[151] National Aeronautics and Space Administration (NASA). Curiosity Rover - Eyes and Other Senses, [Last Access: December 2014]. `http://mars.jpl.nasa.gov/msl/mission/rover/eyesandother/`.

[152] National Aeronautics and Space Administration (NASA). Interface Region Imaging Spectrograph (IRIS) mission, [Last Access: December 2014]. `http://www.nasa.gov/mission_pages/iris/`.

[153] National Aeronautics and Space Administration (NASA). International Space Station, [Last Access: December 2014]. `http://www.nasa.gov/mission_pages/station/`.

[154] National Aeronautics and Space Administration (NASA). Kepler - Spacecraft and Instrument, [Last Access: December 2014]. `http://www.nasa.gov/mission_pages/kepler/spacecraft/`.

[155] National Aeronautics and Space Administration (NASA). Kepler mission, [Last Access: December 2014]. `http://www.nasa.gov/mission_pages/kepler/overview/`.

[156] National Aeronautics and Space Administration (NASA). Lunar Atmosphere and Dust Environment Explorer (LADEE) mission, [Last Access: December 2014]. http://www.nasa.gov/mission_pages/ladee/.

[157] National Aeronautics and Space Administration (NASA). Mars Descent Imager (MARDI), [Last Access: December 2014]. http://mars.jpl.nasa.gov/msl/mission/instruments/cameras/mardi/.

[158] National Aeronautics and Space Administration (NASA). Mast Camera (Mastcam), [Last Access: December 2014]. http://mars.jpl.nasa.gov/msl/mission/instruments/cameras/mastcam/.

[159] National Aeronautics and Space Administration (NASA). Mission to Moon: Ranger 7, [Last Access: December 2014]. http://www.jpl.nasa.gov/missions/ranger-7/.

[160] National Aeronautics and Space Administration (NASA). NEO - NASA Earth Observation, [Last Access: December 2014]. http://neo.sci.gsfc.nasa.gov/.

[161] National Aeronautics and Space Administration (NASA). Rosetta NAVigation CAMera (NAVCAM), [Last Access: December 2014]. http://starbrite.jpl.nasa.gov/pds/viewInstrumentProfile.jsp?INSTRUMENT_ID=NAVCAM&INSTRUMENT_HOST_ID=RO.

[162] National Aeronautics and Space Administration (NASA) - Earth Observatory. Global Albedo, [Last Access: December 2014]. http://visibleearth.nasa.gov/view.php?id=60636.

[163] Trung Tran Ngoc, Phong Vo Dinh, and Bac Le Hoai. Detecting probable regions of humans in still images using raw edges. In *Proc. of International Conference on Knowledge and Systems Engineering (KSE)*, pages 73–80, 2009.

[164] University of Oxford. Affine Covariant Regions Dataset, [Last Access: December 2014]. www.robots.ox.ac.uk/~vgg/data/data-aff.html.

[165] F. Ono, S. Kino, M. Yoshida, and T. Kimura. Bi-level image coding with melcode-comparison of block type code and arithmetic type code. In *Proc. of Global Telecommunications Conference and Exhibition 'Communications Technology for the 1990s and Beyond' (GLOBECOM)*, pages 255–260 vol.1, 1989.

[166] Michael Oren and Shree K. Nayar. Generalization of lambert's reflectance model. In *Proc. of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 239–246, 1994.

[167] Thorsten Overgaard. The Grandfather of 35mm Photography, [Last Access: December 2014]. http://www.overgaard.dk/leica_history.html.

[168] Deuk Hyun Park, Hyoung Seok Ko, Jae Gon Kim, and Jun Dong Cho. Real time rectification using differentially encoded lookup table. In *Proc. of the 5th International Conference on Ubiquitous Information Management and Communication*, page 47. ACM, 2011.

[169] B. V. Pham. Systeme de navigation absolue pour l'atterrissage d'une sonde interplanetaire, 2010.

[170] Regione Piemonte. CApture and DE-orbiting Technologies (CADET) project, [Last Access: December 2014]. http://web.aviospace.com/cadet/index.php/obj.

[171] Tsai Ping-Sing and Mubarak Shah. Shape from shading using linear approximation. *Image and Vision Computing*, 12(8):487–498, 1994.

[172] R. Prakash, P.D. Burkhart, A. Chen, K.A. Comeaux, C.S. Guernsey, D.M. Kipp, L.V. Lorenzoni, G.F. Mendeck, R.W. Powell, T.P. Rivellini, A.M. San Martin, S.W. Sell, A.D. Steltzner, and D.W. Way. Mars science laboratory entry, descent, and landing system overview. In *Proc. of IEEE Aerospace Conference*, pages 1–18, 2008.

[173] Rochit Rajsuman. *System-on-a-Chip: Design and Test*. Artech House, Inc., 1st edition, 2000.

[174] R.W. Redington. Introduction to the vidicon family of tubes. In LucienM. Biberman and Sol Nudelman, editors, *Photoelectronic Imaging Devices*, Optical Physics and Engineering, pages 263–273. Springer US, 1971.

[175] Iain E. Richardson. The h.264 advanced video compression standard, 2nd edition. 2010.

[176] M V Rohith, Scott Sorensen, Stephen Rhein, and Chandra Kambhamettu. Shape from stereo and shading by gradient constrained interpolation. In *Proc. of 20th IEEE International Conference on Image Processing (ICIP)*, pages 2232–2236, 2013.

[177] Francesco Rosso, Francesco Gallo, Walter Allasia, Enrico Licata, Paolo Prinetto, Daniele Rolfo, Pascal Trotta, Alain Favetto, Marco Paleari, and Paolo Ariano. Stereo vision system for capture and removal of space debris. In *Proc. of Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 201–207. IEEE, 2013.

[178] S. Rota Bulo and P. Kontschieder. Neural decision forests for semantic image labelling. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 81–88, 2014.

[179] F. Russo. A method for estimation and filtering of gaussian noise in images. *IEEE Transactions on Instrumentation and Measurement*, 52(4):1148 – 1154, 2003.

[180] Heuel S. 1ma239: Radar waveforms for a&d and automotive radar. *White paper of Rohde & Schwarz*, 2013. http://www.rohde-schwarz.com/en/applications/radar-waveforms-for-a-d-and-automotive-radar-application-note_56280-50249.html.

[181] Micro-Cameras & Space Exploration SA. Digital Space Micro-Cameras, [Last Access: December 2014]. http://www.microcameras.ch/site/index.php?id=48.

[182] Hajar Sadeghi, Payman Moallem, and S Amirhassan Monadjemi. Feature based color stereo matching algorithm using restricted search. In *Proc. of European Computing Conference*, pages 105–111. Springer, 2009.

[183] A.M. Saleh, J.J. Serrano, and J.H. Patel. Reliability of scrubbing recovery-techniques for memory systems. *IEEE Transactions on Reliability*, 39(1):114–122, 1990.

[184] Diego Santa-cruz and Touradj Ebrahimi. A study of jpeg 2000 still image coding versus other standards. In *Proc. of European Signal Processing Conference - Volume 2*, pages 446–454, 2000.

[185] W. Saunders, A.J. McGrath, W. Saunders, and A.J. McGrath. What? a large reflective schmidt telescope for the antarctic plateau. *European Astronomical Society (EAS) Publications Series*, 25:261–264, 2007.

[186] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[187] Michael Smith. *Application-Specific Integrated Circuits*. Addison-Wesley Professional, 1st edition, 1997.

[188] S. M. Smith and J. M. Brady. SUSAN - a new approach to low level image processing. *International Journal of Computer Vision (IJCV)*, 23:45–78, 1995.

[189] SONY. Avchd introduction handbook, 2nd edition. 2009.

[190] S. Sowmya and R. Paily. FPGA implementation of image enhancement algorithms. In *Proc. of 2011 International Conference on Communications and Signal Processing (ICCSP)*, pages 584–588, 2011.

[191] M. Spigai, S. Clerc, and V. Simard-Bilodeau. *Crater detection with segmentation-based image processing algorithm*, 2010.

[192] Changming Sun. Fast algorithm for local statistics calculation for n -dimensional images. *Real-Time Imaging*, 7(6):519 – 527, 2001.

[193] Synopsys. Synplify Pro , [Last Access: December 2014]. http://www.synopsys.com/Tools/Implementation/FPGAImplementation/FPGASynthesis/Pages/SynplifyPro.aspx.

[194] Richard Szeliski. Fast shape from shading. In Olivier D. Faugeras, editor, *Proc. of European Conference on Computer Vision (ECCV)*, volume 427, pages 359–368, 1990.

[195] Tadewos Getahun Tadewos, Paolo Prinetto, Daniele Rolfo, Pascal Trotta, Piergiorgio Lanza, Andrea Martelli, and Antonio Tramutola. A novel approach for Video-Based Absolute Navigation in space exploration missions. In *Proc. of Data Systems In Aerospace Conference (DASIA)*, 2013.

[196] Shen-Chuan Tai and Shih-Ming Yang. A fast method for image noise estimation using laplacian operator and adaptive edge detection. In *Proc. of 3rd International Symposium on Communications, Control and Signal Processing (ISCCSP)*, pages 1077 – 1081, 2008.

[197] Yibing Tang, Xijun Hua, M. Yokomichi, T. Kitazoe, and M. Kono. Stereo disparity perception for monochromatic surface by self-organization neural network. In *Proc. of the 9th International Conference on Neural Information Processing (ICONIP)*, volume 4, pages 1623–1628 vol.4, 2002.

[198] David Taubman. Jpeg2000: Image compression fundamentals, standards and practice (the international series in engineering and computer science). 2013.

[199] Camillo J Taylor. Surface reconstruction from feature based stereo. In *Proc. of 9th IEEE International Conference onComputer Vision*, pages 184–190. IEEE, 2003.

[200] FB ter Haar. *Reconstruction and analysis of shapes from 3D scans*. PhD thesis, Utrecht University, 2009.

[201] Texas Instruments. *FIFO Architecture, Functions, and Applications*, 1999. SCAA042A.

[202] Texas Instruments Inc. *LVC and LV Low-Voltage CMOS Logic*, 1998.

[203] Thales Alenia Space S.p.a. Aerospace module speed and trajectory estimation - internal report. Technical report, 2012.

[204] The Consultative Committee for Space Data System. Image data compression - recommended standard - ccsds 122.0-b-1. 2005.

[205] The U.S. Geological Survey USGS Water Science School. How much water is there on, in, and above the Earth, [Last Access: December 2014]. http://water.usgs.gov/edu/earthhowmuch.html.

[206] S Thomas, K Papadimitriou, and A Dollas. Architecture and implementation of real-time 3D stereo vision on a Xilinx FPGA. In *Proc. of IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 186–191. IEEE, 2013.

[207] Jing Tian and Li Chen. Image noise estimation using a variation-adaptive evolutionary approach. *IEEE Signal Processing Letters*, 19(7):395 – 398, 2012.

[208] V. Torre and T. Poggio. *Differential Operators for Edge Detection - WP-252*, 1983.

[209] Stephen Trimberger. *Field-Programmable Gate Array Technology*. Springer Science and Business Media, 1994.

[210] G. Troglio, J. Le Moigne, J.A. Benediktsson, G. Moser, and S.B. Serpico. Automatic extraction of ellipsoidal features for planetary image registration. *IEEE Geoscience and Remote Sensing Letters*, 9(1):95 –99, 2012.

[211] Tinne Tuytelaars and Krystian Mikolajczyk. *Local Invariant Feature Detectors: A Survey*. Now Publishers Inc., 2008.

[212] Cristian Vancea and Sergiu Nedevschi. LUT-based image rectification module implemented in FPGA. In *Proc. of IEEE International Conference on Intelligent Computer Communication and Processing*, pages 147–154. IEEE, 2007.

[213] V.N. Varghees, M.S. Manikandan, and R. Gini. Adaptive MRI image denoising using total-variation and local noise estimation. In *Proc. of International Conference on Advances in Engineering, Science and Management (ICAESM)*, pages 506 – 511, 2012.

[214] C.Y. Villalpando, R.A Werner, J.M. Carson, G. Khanoyan, R.A Stern, and N. Trawny. A hybrid FPGA/Tilera compute element for autonomous hazard detection and navigation. In *Proc. of IEEE Aerospace Conference*, pages 1–9, 2013.

[215] Fan Wang and V.D. Agrawal. Single event upset: An embedded tutorial. In *Proc. of 21st International Conference on VLSI Design (VLSID)*, pages 429–434, 2008.

[216] Shoujia Wang, Wenhui Li, Ying Wang, Yuanyuan Jiang, Shan Jiang, and Ruilin Zhao. An improved difference of gaussian filter in face recognition. *Journal of Multimedia*, 7(6), 2012.

[217] Wenqiang Wang, Jing Yan, Ningyi Xu, Yu Wang, and Feng-Hsiung Hsu. Real-time high-quality stereo vision system in FPGA. In *Proc. of International Conference on Field-Programmable Technology (FPT)*, pages 358–361. IEEE, 2013.

[218] Ze Wang, Tianxu Zhang, Luxin Yan, and Cheng Gong. A high performance fully pipelined architecture for lossless compression of satellite image. In *Proc. on International Conference on Multimedia Technology (ICMT)*, pages 1–4, 2010.

[219] M. J. Weinberger, G. Seroussi, and G. Sapiro. The loco-i lossless image compression algorithm: Principles and standardization into jpeg-ls. *IEEE Transaction on Image Processing*, 9(8):1309–1324, 2000.

[220] Wikipedia. Image file formats, [Last Access: December 2014]. `http://en.wikipedia.org/wiki/Image_file_formats`.

[221] K. Worners, R. Le Letty, L. Summerer, R. Schonenborg, O. Dubois-Matra, E. Luraschi, A. Cropp, H. Krag, and J. Delaval. ESA Technologies for Space Debris remedation, [Last Access: December 2014]. `http://www.esa.int/gsp/ACT/doc/MAD/pub/ACT-RPR-MAD-2013-04-KW-CleanSpace-ADR.pdf`.

[222] Qiang Wu, Yaobin Chi, and Zhiyong Wang. CCD noise effect on data transmission efficiency of onboard lossless-compressed remote sensing images. In *Proc. of International Conference on Information Engineering and Computer Science (ICIECS)*, pages 1 – 4, 2009.

[223] Xiaolin Wu, Nasir Memon, and Khalid Sayood. A context-based, adaptive, lossless/nearly-lossless coding scheme for continuous-tone images. In *ISO/IEC JTC 1/SC 29/WC 1 document No*, 1995.

[224] Ning Xie and A.J.P. Theuwissen. An autonomous microdigital sun sensor by a cmos imager in space application. *IEEE Transactions on Electron Devices*, 59(12):3405–3410, 2012.

[225] Xilinx Inc. *Space-Grade Virtex-4QV Family Overview*, 2010. Ver. 2.0 , DS653.

[226] Xilinx Inc. *Virtex-4 Family Overview*, 2010. Ver. 3.1 , DS112.

[227] Xilinx Inc. *LogiCORE IP Soft Error Mitigation Controller v3.4* , 2012. PG036 Product Guide.

[228] Xilinx Inc. *Radiation-Hardened, Space-Grade Virtex-5QV Family Overview*, 2012. Ver. 1.3, DS192.

[229] Xilinx Inc. *ISE Design Suite 14: Release Notes, Installation, and Licensing*, 2013. Ver. 14.6 , UG631.

[230] Xilinx Inc. *Partial Reconfiguration User Guide*, 2014. Ver. 14.5 , UG702.

[231] Xilinx Inc. *Virtex-II Platform FPGAs: Complete Data Sheet*, 2014. Ver. 4.0 , DS031.

[232] Sudhakar Yalamanchili. *Introductory VHDL: From Simulation to Synthesis*. Prentice Hall PTR, 1st edition, 2000.

[233] Lifan Yao, Hao Feng, Yiqun Zhu, Zhiguo Jiang, Danpei Zhao, and Wenquan Feng. An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher. In *Proc. of International Conference on Field-Programmable Technology (FPT)*, pages 30–37, 2009.

[234] Guoxia Yu, Tanya Vladimirova, and Martin Sweeting. An efficient on-board lossless compression design for remote sensing image data. In *Proc. of IEEE International Geoscience & Remote Sensing Symposium (IGARSS)*, pages 183–190, 2008.

[235] Guoxia Yu, Tanya Vladimirova, Xiaofeng Wu, and M.N. Sweeting. A new high-level reconfigurable lossless image compression system for space applications. In *Proc. of NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 183–190, 2008.

[236] K. Zaitsu, K. Tatsumura, M. Matsumoto, M. Oda, S. Fujita, and S. Yasuda. Flash-based nonvolatile programmable switch for low-power and high-speed FPGA by adjacent integration of MONOS/logic and novel programming scheme. In *Proc. of Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers*, pages 1–2, 2014.

[237] Ming Zeng, Ting Yang, Youfu Li, Qinghao Meng, Jian Liu, and Tiemao Han. Finding regions of interest based on scale-space keypoint detection. In Mark Zhou and Honghua Tan, editors, *Advances in Computer Science and Education Applications*, volume 202 of *Communications in Computer and Information Science*, pages 428–435. Springer Berlin Heidelberg, 2011.

[238] Ruo Zhang, P-S Tsai, James Edwin Cryer, and Mubarak Shah. Shape-from-shading: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706, 1999.

[239] Xiaozheng Zhang and Yongsheng Gao. Generalised ambient reflection models for lambertian and phong surfaces. In *Proc. of the 16th IEEE International Conference on Image Processing*, pages 3945–3948, 2009.

[240] Feng Zhao, Qingming Huang, and Wen Gao. Image matching by normalized cross-correlation. In *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 2, pages II–II, 2006.

[241] Feng Zhao, Qingming Huang, and Wen Gao. Image matching by normalized cross-correlation. In *Proc. of International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages II:729–732, 2006.

[242] Zhen-Bing Zhao, Jin-Sha Yuan, Qiang Gao, and Ying-Hui Kong. Wavelet image de-noising method based on noise standard deviation estimation. In *Proc. of International Conference on Wavelet Analysis and Pattern Recognition (ICWAPR)*, volume 4, pages 1910 – 1914, 2007.

[243] P Zicari, H Lam, and A George. Reconfigurable computing architecture for accurate disparity map calculation in real-time stereo vision. pages 3–10, 2003.

[244] Nicholas Zinner, Austin Williamson, Kristen Brenner, John Benjamin Curran, Ahna Isaak, Matthew Knoch, Adam Leppek, and Jenae Lestishen. Junk hunter: Autonomous rendezvous, capture, and de-orbit of orbital debris. *Proc. of AIAA SPACE Conference & Exposition*, 2011.

[245] Chengming Zou and E. Hancock. Recovering 3D shape using an improved fast marching method. In *Proc. of 20th International Conference on Pattern Recognition (ICPR)*, pages 1678–1681, 2010.